USE OF LOG DATA OF INFORMATION SYSTEMS IN SUPPORTING DECISION MAKING

Pekka Sallinen

University of Tampere
School of Information Science
Information Studies and
Interactive Media
Master's Thesis
May 2015

UNIVERSITY OF TAMPERE, School of Information Science
Information Studies and Interactive Media
SALLINEN, PEKKA: Use of Log Data of Information Systems in Supporting Decision Making
Master's Thesis, 58 p., Appendices 12 p.
May 2015

Abstract
This thesis introduces a study of how log data from an information system run as a service can be utilized to support decision making in various work roles in the administration and management of the service. In this study, I interviewed persons working in these roles to determine their information needs and to gather requirements for a log data based dashboard system, built on a commercial log data indexing and search tool called Splunk. While alerting and reporting of technical problems in running the service is important, the aim of the thesis is to seek other uses for the log data, such as reporting process and service status and developments in a way that could be utilized by the management and in other non-technical decision maker roles around the service.

The interviews are reviewed to determine what information to include in the information search and the dashboards used for the visualizations of the data. The information system is then created based on analysis these interviews, and the system and its usability for the purpose is evaluated.

For creation of the system both actual log data and data from operational SQL databases was used. For gathering the required data from SQL database for analysis in Splunk a pseudo-log was created. A Java Servlet paired with a Python script was programmed to circumvent some access restrictions to the database and to fetch the data. Problems encountered during the implementation process are described, and ways to fix them are suggested. These problems partly reduced the usability of the information gathered, but some important lessons were learned in the process, especially about the importance of source data validation and verification.

Keywords: Logs, metrics, decision making, decision support systems, Splunk

Table of Contents

1 INTRODUCTION

Creating reports and metrics for superiors in a large company is often a time consuming operation. It is common that data is gathered manually from various sources, then entered into Excel sheets for analysis, after which the numbers, possibly with graphical representation, are transferred to a PowerPoint slide set for presentation[1]. Some metrics and reporting requests may be communicated to the analyst, but often it is somewhat unclear what should be metered and reported in the first place. Before reports can be created automatically these questions need to be studied.

In this thesis I study the basic requirements for metrics and reporting of an information service, and aim to create an automated system to produce such metrics and reports in near-real-time manner.

The data utilized in the analysis is mainly *log data*, which is an interesting subject in itself. Log data is often neglected as an information source, although automatically created by the running information system. This differs from the traditional approach to reporting, which aims to build the reporting functionality into the information system itself. An integrated reporting interface requires much effort in development phase of the information system, and may be difficult to alter if reporting requirements are changed, while analyzing the log data can be done any time and with different tools if necessary. In recent years, some analysis engines have been launched for log data, making it much easier to build such reporting interfaces utilizing log data. In this thesis, implementation is relying quite heavily on Splunk®, which can currently be considered the market leader in the log analysis field.

The approach in this thesis is constructive. The main research question is: How should the log data be analyzed and represented to support decision making in line management and

[1] Microsoft corporation is working on tools for this kind of analysis, calling the offering "Power BI". While Excel still has very important role in the Power BI application(s) by Microsoft, tools for gathering data automatically from databases etc. are under development, but seem to lack some maturity and features as of May 2015. In an organization that relies on Microsoft tools and may already have collected data in Excel workbooks Microsoft Power BI is well worth consideration.

management levels over the line management level? Line management is working in close co-operation with the personnel, so log data can be assumed to contain relevant information for making decisions on the upkeep of and investments on the information systems.

A secondary research question is to gain insights into what kinds of decisions line managers have to make, what kinds of information sources they prefer, and how they provide information to his/her superiors. These questions are closely related to the main research question.

The thesis is structured as follows: I will first take a look in general concepts of log data, decision making and managerial work. I will then study how log data analysis and reporting based on log data analysis can support managers in their decision making, and what are the limitations of this system. The requirements for the system are clarified by interviews of managers and metrics specialist(s) in an industrial company in high technology field.

2 THEORY AND GENERAL CONCEPTS

2.1 Information system as a service

In this thesis, the emphasis is in *services,* where the information system upkeep is separated from the user so that the user does not need to worry about the resourcing or internals of the information system being used. There are different ways to implement such systems. For example in case of a signing service, a package, typically a software installation package, is submitted for signing (service request), and after processing a signed package is returned to the customer. In case of a software source code management system, the service is running all the time and providing the information stored in the system, resembling a platform or a network drive. In the SAAS model, full software is provided as a service, for example word processor running in a network cloud, saving the user the burden of installing and maintaining the software.

In this thesis the main emphasis is in a signing service, where the process starts with a service request, and ends with a successful signing of the package the signing was requested for. Other services are not analyzed, but the service managers are interviewed to get more information about monitoring needs in general. It should be noted that monitoring other types of services may differ in details, and what to meter should be analyzed separately in different services.

2.2 Log files

Large-scale programs or software systems often keep what are called “log files” - continuous reports of events that have occurred during the running of the program. These log files are typically used for debugging, fault location in production software, regression testing, or administrative information. (Andrews, 2001) Log data of information systems is not a new invention, but has been created almost as long as information systems have existed; first versions being maybe printed in paper and only studied when severe problems occurred.

One characteristic for log data is that it has a strong connection with time. Every event written to the log is stored with a timestamp, recording exactly when the event has happened. This makes it comparably easy to create a time series of different events with proper tools and

create time-based statistics of events happened. Trend lines, time charts and various visualizations can be then created from this data.

Another feature of log data is that it is not altered after it has been written; only new lines with a new timestamp can be written. A log is therefore a static series of records (usually lines), ordered by time of the event recorded. In this thesis, all data that is ordered as a log (file) is treated as log data, the emphasis being on the analysis of such event-based data. Talking about log "files" may be misleading, as the log does not necessarily have to be a file per se, but can reside in a database or other such storage system. Representation of the log data is usually plain text - ASCII and its derivatives – but the storage may also be binary formatted, like in case of Microsoft Windows Event Log, where a special interpretation interface is required for making the data human readable. In this thesis the data is assumed interpreted, and all operations and analysis are performed on data that is human readable - alas not necessarily comprehensible or meaningful as such.

Definition: Log data is a collection of information about events, identified by time of the event, and written into static storage, usually a log file.

The emphasis of this thesis is on utilizing log data as defined above. However, I will use other information sources when necessary to fulfill requirements identified during the process. Finding alternative information sources can also be seen here as a way to define limits for the utilization of the log data. Such practice is also supported by the log analyzing software Splunk, as it has access modules for accessing SQL databases and can even be fitted with a custom interface program. Even then, there is a strong emphasis on the time of the events.

Log data usually has several uses. For example, a log file can be seen as an archive, and storing information in a sense of Schellenberg (1956) having primary value and secondary value. Historically log data have been created mainly for making it easier to detect computer program errors and track those errors and program execution while running, without disturbing the running system. This can be seen as primary value. Among secondary values (in sense of Schellenberg) is evidence, for example which users have used the system at what time. Another is information: Mere listing of events happened can tell us many things, such as how many times something happened in a time frame, or if certain things have happened or not. Such information may indicate various things affecting the decision making.

While log files studied in this thesis are from computer related services, the methods could be applied to other services. Services even in this thesis can essentially be seen as *case processing systems.*

2.3 Decisions

According to Jonassen, decision making is the most common kind of problem solving. It is also an important component in other more ill-structured and complex kinds of problem solving, including policy problems and design problems (Jonassen, 2012). Jonassen classifies decisions to choices, acceptances, evaluations, and constructions, among others. Decision-making can be divided to *normative* (rational) and *descriptive (*naturalistic) approaches. Normative decision-making relies on analysis methods such as decision matrices, SWOT and force field analyses. In contrast, naturalistic approach relies on constructing stories, mental simulations, scenarios, and arguments, emphasizing the meanings of decision options and the role of unconscious emotions in decision making. (Jonassen, 2012)

Jonassen cites Yates and Tschirhart (2006) to list various types of decisions:

- *Choices* - Where you select a subset from larger set of alternatives
- *Acceptances*/*rejections* - A binary choice in which only one specific option is accepted or not
- *Evaluations* - Statements of worth that are backed up with commitments to act
- *Constructions* - Attempts to create ideal solutions given available resources. These complex problems entail multiple decisions.

(Jonassen, 2012)

Churchman (1971) (according to Olson, 2001) identified rational (Leibnizian), empirical (Lockean), multi-perspective frameworks (Kantian), dialectic (Hegelian), and cause-and-effect (Singerian) inquiring systems as alternatives to the rational system.

Olson argues that the rational Leibnizian system is not suitable for information systems except for analyses of specific, well-defined problems. Information systems would play a key role in implementation of the Lockean model of empirical theory building, by providing reports

and data for querying. The Kantian model emphasizes using multiple views, which shared decision-making would provide.

For issues of particular importance, the Hegelian approach of pitting one advocate against another may be a useful way to make quick considerations about important factors involved in a decision. The Singerian model's focus on measurement would be useful in monitoring the impact of actions taken, thus enabling more accurate development of system understanding. (Olson, 2001)

The rational approach (Descartes 1955; Leibniz, 1953, according to Olson, 2001) to decision making is based on the idea of identifying truth through reason, based on a priori deduction. This approach begins with constructing a model of a problem, which is then solved to identify the best solution. (Olson, 2001) According to Olson, this is often successful in closed environments, but rarely in organizations, which are generally open systems, dealing with customers or other entities. Open systems involve high levels of uncertainty and incomplete understanding, along with incomplete and imperfect data. Managerial judgment needs to be applied when high levels of uncertainty, incomplete understanding, and imperfect data are the case. (Olson, 2001, p. 240)

According to Olson, Wildavsky (1997) noted the high levels of uncertainty that decision making involves. We may not even be confident of our own preferences, as they depend on a complete understanding of the effects of our actions. It seems safe to assume that without complete knowledge optimization is not possible, or at least the results may be hard to predict or even chaotic[2].

According to (Olson, 2001), MacCrimmon and Wehrung (1988) published a detailed study of things that real executives do to cope with difficult tradeoffs, again at variance with the rational normative view. Executives were found to have a different aversion to risk, depending upon if gains or losses were at stake. They modified the risk through information gathering, bargaining, delay, and delegation. They also did not settle for choices presented to them, but sought to reframe decision problems by creating superior alternatives. Thus, real decision makers were found to operate in an environment settling for *the best information they*

[2] The term "chaotic" used here in mathematical sense.

could get, realizing that the cost of gathering complete information was too high, or the time to gather it unavailable.(Olson, 2001)

2.4 Classification of decision levels

According to Hätönen, Simon (1960) classifies decisions in three categories or levels, Hätönen describing this approach as traditional:

- Strategic level
- Tactical level
- Automatic level

In a strategic level, the dominating factors of decision making are strategy and vision that are always formulated by man. Vision defines the ultimate goal, while strategy describes the procedure how to reach for the goal. Tactical level includes decisions about concrete action, and at the automatic level the actions are executed and results are monitored and controlled. According to Hätönen this classification is a continuum and not discrete in nature. (Hätönen, 2009, p. 43)

This classification by Simon can be seen as hierarchical, and it may well be in line with the business decision culture even today, especially in large companies such as Nokia or in traditional businesses. It seems to match the ideal of the rational decision making, but does not necessarily translate well to analysis of information needs. It is unclear if monitoring and control of the actions executed provides enough information about the situation to reformulate visions and strategy, or only about the performance of chosen tactics.

2.5 Managerial information needs and information complexity

Ackoff (1967) divides managerial decisions by their information needs into three types:

a) Decisions for which adequate models are available or can be constructed and from which optimal solutions can be derived. These can be seen as fully rational decisions, and decision model describes what information is relevant. Process can be easily automated.

b) Decisions for which adequate models can be constructed but from which optimal solutions cannot be extracted. Here some kind of heuristic or search procedure should be provided even if it consists of no more than computerized trial and error – permitting comparison of proposed alternative results.
c) Decision for which adequate models cannot be constructed. Research is required, or judgment must be used to "guess" what information is relevant. (Ackoff, 1967)

Järvelin (1987) has classified the decisions further by the complexity of the task to five different levels:

1) Automatic information/data processing situation, where the task, input and output can be defined beforehand
2) Normal information processing situation, where the problem is fairly well defined beforehand, but requires some judgment
3) Normal decision making situation, where judgment by the situation has a strong role in both information gathering and processing
4) Known, true decision making situation, where the subject of the decision is known, but there are no pre-defined procedures for making the decision
5) Completely true decision making situation that requires novelty; the situation could not have been foreseen, and no preparations have been possible. In the beginning it is unclear what kind of decision should be made, or should any decision be made.

(Järvelin, 1987)

It is noteworthy that both Ackoff and Järvelin identify the automation of the "easiest level" of the decisions, as well as the need for judgment in "higher" level of decision making. It is obvious that in the higher level of decision, there are gaps in the knowledge of the decision maker, and that to gain that knowledge the decision maker must go through some cognitive process.

Byström (Byström, 1999) reduced the classification further to three: Automatic Information Processing Task (AIPT), Normal Information Processing Task (NIPT), and Decision Task (DT), creating one joint class for both Normal Decision tasks and Genuine Decision Tasks. (Byström, 1999, 70, 80) This classification is also used in this study. The

Decision Task group was considered to be more complex than the other two, which leads to different use of information sources. This study utilizes this classification in the requirements gathering phase by formulating an interview structure, or questions, in a manner that the requirements recognized could be more easily classified into this model. Assumption in this model would be that requirements with lower complexity would be easier to implement, while higher complexity requirements require more analysis before implementation.

2.6 Information needs and limitations of a managerial decision maker

According to Sauter, Mintzberg (1990) found out that most decision makers want to operate their way because it has been successful for them. As a result, a decision support system must also be designed to allow their users to *do things their way.* In other words, they must include substantial flexibility in their operations.(Sauter, 2011, p. 31)

Herbert Simon (among others) also notified that there is a limit for how much information an individual can process. Therefore "more information" is not equivalent to "as much information as possible". Russel Ackoff argued that *the last thing a manager needs was more information:*

> *Most MIS's[3] are designed on the assumption that the critical deficiency under which most managers operate is the lack of relevant information. I do not deny that most managers lack a good deal of information that they should have, but I do deny that this is the most important informational deficiency from which they suffer. It seems to me that they suffer more from an over abundance[sic] of irrelevant information. (Ackoff, 1967, p. 147)*

This indicates that while essential information should be provided, extra effort should be put into avoiding information overflow.

Ackoff also notes that it must be assumed that the system that is being designed will be deficient in many and significant ways. Therefore it is necessary to identify the ways in which it may be deficient, to design procedures for detecting its deficiencies, and for correcting the system so

[3] Management Information System

as to remove or reduce them. Hence the system should be designed to be flexible and adaptive. (Ackoff, 1967, p. B-155)

Based on these observations and my personal experience, I present three basic rules:

1) Keep it simple (and easy)
2) Stick to the essential information (to avoid information overflow)
3) Offer flexibility

2.7 Relation of information sources and task complexity

Byström found evidence about connection between types of information and information sources. The role of people as sources was strengthened in tasks where both task and domain information was required, experts inside the organization being the most consulted sources, followed by meetings. (Byström, 1999, p. 96)

Also, task complexity was found to have a connection to information and information sources used. The more complex the task, the more information types were needed and thus, the more useful to several tasks the information acquired was. The *use of people as sources* also increased with task complexity, being "overwhelmingly more popular than documentary sources in DTs". In addition, the internality[4] increased with task complexity. (Byström, 1999, pp. 98–99,101–102).

It is also notable that, according to Byström, the experts and meetings were extremely useful sources of all information types (task, domain and task-solving information) (Byström, 1999, p. 109), being therefore useful in all kinds of tasks requiring information processing. Therefore, it can be assumed that persons successful in processing and handling large amounts of information have learned to use people as their main source of information, and therefore tend to do this even more. This is also in line with findings by Mintzberg.

Based on the findings it can be assumed that it is natural for someone successful as a manager to leave more simple tasks (AIPT, NIPT) to people with specific knowledge and concentrate on more complex tasks, probably concerning several organizational units.

[4] Internal = inside same organization as the task performer

Therefore there is limited need for detailed information about any single problem domain, and even if there is such a need, the information can be acquired from people having special knowledge, rather than documentary sources or databases, which would most probably require cognitive involvement in the problem domain. According to Byström it is commonly stated in media richness literature that text based media are connected with routine tasks, whereas media that appear more flexible (e.g., face-to-face, video conferencing and telephone) are better suited to complex tasks. High task complexity – almost in spite of the information types required – is best managed with flexible sources.

It is also noted by Mintzberg that managers have little time to probe complex issues, and therefore rely on expert analysts on providing information – and action plans - they need for various situations. (Mintzberg, 1990, p. 175) It seems reasonable to assume that by use of other people and their expertise the decision maker is able to circumvent the need for excessive cognitive processing, thus making it easier to gather essential information for the decision problem at hand.

Based on this we can assume that in more complex decision problems the information system being built is not the preferred information source, at least directly. In most complex problems the manager having the decision task at hand goes to other people, who should have the information needed from the system.

The system therefore has two separate informational functions:

1) The system itself being an information source for the decision maker.
2) The person using the system being the information source for the decision maker.

These two use cases shall be taken into account when gathering the requirements.

3 METHODOLOGY

3.1 Design research

According to Järvinen (2012), in design research we try to answer the question: Can we build a certain innovation and how useful is a particular innovation. We can also ask, which kind a certain innovation ought to be, and how ought we to build a particular innovation. The purpose of design research is to achieve a permanent change in the system, from the initial state to the desired one. (Järvinen, 2012, p. 98) According to March & Smith (1995) and later (according to Järvinen) Hevner et al (2004), design science consists of two basic activities, build and evaluate. Building is a process of constructing an artifact/innovation for a specific purpose; evaluation is a process of determining how well the artifact performs. (Järvinen, 2012, p. 100)

Järvinen cites Peffers et al. (2007) in methodology of six activities:

1) Problem identification and motivation
2) Define the objectives for a solution
3) Design and development
4) Demonstration
5) Evaluation
6) Communication

First, the preferred goal state is to be defined. Data for this phase is collected by interviewing managers and possibly other specialists in the field. From interviews information needs will be recognized. After the evaluation of specific information needs, the potential candidates for implementation will be selected.

This study therefore contains two (or three) phases:

1) The specification process
2) The implementation process

The third possible process would be *evaluation of the implementation*

To gather requirements for the system potential users of the system are interviewed. The most relevant person for this study is the line manager responsible of the system creating the logs

and the team that is managing the system. He is the person who decides or approves any changes to the system and who should be aware of any problems or developments of interest in usage of the system. Other team members are also to be interviewed, for example the service manager, responsible of arranging upkeep and seeing everything is running smoothly on daily basis. To screen metrics and decision making principles in the company in general, a specialist with experience in these areas is also interviewed.

Themes are prepared beforehand to steer the interview, but some specific questions are also prepared beforehand based on the prior knowledge about decision-making in general and decision support systems. The manager's preferred methods of information gathering are determined to assess how much information is actually reasonable to produce from the information system being built.

3.2 Data gathering specification process by interviews

The most typical data gathering methods are interview, observation, questionnaire and written material. In the interview, there are the options of either a deep interview or a theme interview; observation can be participant or non-participant; in questionnaire there can be structured and open questions, and in written material you can find descriptions and decisions. (Järvinen, 2012, p. 140).

In the building case of constructive approach, a developer of the new artifact is not only interviewing prospective users to find out their requirements, but also telling the opportunities he/she sees concerning the new artifact. In discussion between two experts, the developer and the user, both must cross-educate one another to understand potential opportunities and restrictions in building and future use of the new artifact. (Järvinen, 2012, p. 140)

The requirements for this information system are gathered by interviewing the end users, in this case mainly line managers. The purpose for the interview is to recognize information needs of both the line manager and his/her superiors, as it is assumed that higher level managers gather information by requesting it from the line manager.

In the interview the main target is to determine requirements for the system. As described before in section "Relation of information sources and task complexity", according to Järvelin

and Byström, information needs of routine tasks are easiest to automate, while complexity of the task at hand makes information gathering more difficult. Therefore it can be assumed that it is easier to implement system with basic information needed for easier decision tasks.

In the interview I will use following classification for the questions:

1) Determine routine tasks and their information needs
2) Determine recurring information needs
3) Determine frequent or probable information queries form upper management
4) Determine failure/problem situations requiring immediate manager actions

3.3 Analysis of the interviews and identification of requirements

In this section I will analyze the interviews and identify needs and specific requirements for the system being built.

As described before, the information needs are quite different in different tasks, complicated tasks requiring wider selection of information sources and possibly interpersonal activities to be performed effectively. Therefore we need to identify

- Which information needs can be served directly by the system (easy tasks)
- Which persons should be provided with which tools to provide enough information

According to Järvinen the *state-transition heuristics* means that the original problem definition, i.e., the initial state, is transformed to the goal state in the sequence of transitions. A research can report every transition from state i to state i+1. If there are too many transitions to be included into a report, the most significant transformations can be described. (Järvinen, 2012, p. 107) In this study I am interested in supporting decision maker in decision making situations. If we assume that there is a collection of such decisions, all more or less different and requiring different information, the problem can be treated so that each information need – or problem - recognized can be solved independently by creating a search and a visualization of the results.

3.4 Interviews

To determine the key features for the log reporting six persons were interviewed. Among interviewees were: two Service Managers, two Senior Solution Delivery Managers, one Director (manager for the abovementioned Sr. Solution Delivery Managers), and one Signing Specialist, who had history in creating metrics to support decision making in his previous work position. All interviewees are working in same organization and sharing same business decision processes. The business decision process itself is not in the scope of this thesis, concentrating on the service level decision-making.

I label the interviewees as follows:

- **Service Manager 1 (SM1):** The service manager for the signing services, responsible for the actual systems where the implementation will take place
- **Senior Solution Delivery Manager 1 (SDM1):** The manager for SM1, and the first manager level to receive reports of the system
- **Director (DIR):** The manager for SDM1, receives reports from SDM1.
- **Service Manager 2 (SM2):** The service manager in other service based internal team in same company, sharing organizational structure and idea with the signing team.
- **Sr. Solution Delivery Manager 2 (SDM2):** Manager of the SM2, quite similar to SDM1.
- **Product Specialist (PS)** was interviewed mainly for his experience in creating metrics to backup various product decision processes.

The questions, or rather themes, were formulated to determine who would actually be interested in information based on service usage data, and if there actually was any need for this information in the interviewees, scope of work, especially in decision making.

To refine requirements for the log data analysis I shall first screen the SDM1 interview and the SM1 interview, as these are most relevant for the implementation project. Other interviews are then screened to further refine the overall information model for the system.

Most of the interviews were performed in Finnish language, as that is the native language of the writer and most of the interviewees. Senior Solution Manager 1 was interviewed in English. The interviews or their transcriptions have not been translated.

3.5 Summary of the interviews and findings

In this section I will first summarize every interview and make notion of the main points of the respective interview. From these summaries a collection of concrete requirements and restrictions for the system being built can then be constructed. I also try to address problems identified in the interviews not directly related to the information system being built and find solutions that can be implemented in human interaction and work practices of the team responsible of the service.

3.5.1 Senior Solution Delivery Manager 1

SSDM1 defines his work as 40% as routine and 60% really dealing with data and information. Main challenge in SSDM1 work concerning information is that the meaning of reporting and gathering information, or the use of the information, is not always clear. This defines a need for motivation, or a need to know why something is done. If gathering and sharing some information has no motivation or the motivation is poorly communicated, a person may feel that the task is unimportant and postpone or even abandon the task altogether.

According to the interviewee the service related information content gathered by the SSDM1 is considerably reduced when reports to the next level manager are produced. Status of the service and service management is represented by a traffic light that can be interpreted as a signal for need of action. Green signals there is no need for action, while yellow and red are used to communicate need for change and problems. The interviewee also noted that often monitoring the data does not lead to any action if everything is fine. Only if the data available indicates that something has changed in a way that requires action an action is initiated. SSDM1 emphasizes that the decision to not to act is a decision too.

SSDM1 describes the service work as routine, but considers service enhancements as creative work.

About gathering information SSDM1 recognizes two aspects of information: the raw data which is used as basis for decisions, and parameters or limits imposed by the organization and management. There is, for example, limit on the amount of money that can be used when resources are required to solve a problem implicated by the data gathered.

A clear preference to use information systems like Google and company intranet was noticed, while asking help from people is preferred in non-technical issues. Meetings were also mentioned as an information source.

Quite interesting fact in SSDM1 interview was that information requests from upper management did not usually require information seeking activities, but SSDM1 felt that he had the answers in hand when needed. Information seeking activities were more often initiated by – for example - external parties that are not familiar with processes and actions of the unit being asked. In case a very specific technical question is being asked forwarding the question to a specialist of that field is preferred by the SSDM1.

About decision making SSDM1 states that while often the actual decisions are not made by SSDM1 personally, he is still proposing the decision and the decisions being made are based on the proposal of SSDM1. This is mostly due to limits imposed by the organization, and any decision beyond these limits requires at least approval from a manager with wider authority. In this study I will treat these proposals and the decisions as equals, as all the low-level information gathering is done while preparing the proposal, the actual decision being always based on this proposal. This is similar to Byström's study in municipal decision making environment (Byström, 1999).

According to the interviewee, the initiator for decision-making is related to need. This need can be a business need, like a new use case or an enhancement, or change in external environment. A vendor change is named as an example of an external trigger that requires a decision to be made. Identifying that something that would be needed is missing is a trigger, or request for change from some instance like upper management or some other party, like security forum. Separation between internal and external triggers is made by SSDM1, with a notion that "quite often it is that something changes, externally".

Use of Key Performance Indicators (KPI) in reporting is identified as strongly related to outsourcing where they are important basis for invoicing; worse performance meaning the outsourcing company can charge less for the service. Therefore the KPI subset in use in internal team can be seen as an informative tool and actually supporting decisions in a less concrete manner. For outsourcing purposes KPIs are defined in an agreement between the companies, while in internal teams the indicators are less formal and can be changed quite flexibly if needed.

According to the interviewee, KPIs in use are monitored to check whether the service is running within acceptable parameters and according to expectation. These parameters and expectations however are not defined accurately anywhere, but are rather the manager's own perceptions about what should be happening. A decision is then made if everything is fine or if some action is required.

It is identified in the interview that historical perspective is important while evaluating the performance indicators and whether the values for KPIs are acceptable. The interviewee mentions that a four-month reference period gives "some idea" of how the service is running.

Another KPI is mentioned to give some idea of how much the service or a service component costs for the company[5]. This information is then used to plan the details of the contracts from certain providers, providing some part of the service in question. Third aspect of KPIs is reporting to higher levels of the organization.

An answer to a follow-up question about parameters and boundaries reveals that while KPI:s are often defined as averages - like average time used for service request process – there are deviations that should be noticed while average values are within acceptable boundaries. There can also be special needs which require different monitoring to ensure that the service is running within acceptable boundaries. Special cases that do not follow normal, defined processes should be monitored more closely. If the workflow is changed for some special reason the exception should be monitored.

When asked about preferred frequency for the reporting, monthly reporting is seen as adequate. A real-time information system would be an improvement. No situations were identified from the past that would have required such an information system. It was identified that for specific technical problems an alerting system should be in place.

For reporting items it was recognized that the system provides some specific information about the status of some development processes that is of interest to wider audience than just the team or department running the service. In this sense the service usage by a certain customer, or

[5] Variable costs

certain use case, provides information about the situation and needs of the customer. This information can be used to plan resourcing, like if the computer hardware capability is adequate for the use case or the customer. Increased usage of the service, especially if originating from a specific customer, is also seen as a signal that indicates the need to operate at a higher level of readiness. Monitoring usage of the service originating from users from different companies is also named as an item requiring monitoring.

For situations needing immediate action, or alert for existence of such situation, it is mentioned that risky or not-so-normal arrangements in service that can be misused should be followed more closely, to ensure that the arrangement is used properly.

For historical data it is only mentioned that the cost causing features should be monitored, but just to get an idea how the things are developing; according to the interviewee there is no need for accuracy, ballpark type of estimate is accurate enough. Wide variation in one metric – service request processing time – is also mentioned as a possible indicator of a problem and need for fixing something proactively, before the problem is starting to cause significant effects. Preferred visualization is *a trend line*, where one can quickly get an idea how things are developing.

3.5.2 Service Manager 1

For Service Manager 1 the work was identified as half routine, half more challenging or complex. The main challenge of SM1 was to get enough information from production systems (servers) due to hosting arrangements being such that information has to be requested from hosting organization. SM1 mentions that logging wrong information, doing wrong or inaccurate analysis and not creating fine (as in coarse / fine) enough logs for analysis.

For legacy systems it was also identified that too much information is logged. It was also noted that this problem can be addressed with suitable tool for searching the logs, in this case Splunk.

Similar to SSDM1, the SM1 also makes note that monitoring old, stable services and their environment is mostly routine work, whereas new services that are constantly being developed is more challenging and complex. According to the interview it should be verified that the new version developed and implemented works as well as the previous version, and preferably better.

One specific problem identified by the SM1 is the lack of information about service users. According to SM1, there is currently no information available about how many users – if any – there are from external organizations[6]. According to interview the problem is that user information resides on different system (LDAP[7]) than the service itself, so there may be a need to implement some separate mechanism for acquiring user information related to service usage. Programming such a tool with Python programming language is mentioned as a possible solution for this, possibly because Splunk tool being used has a built-in Python interpreter/compiler that could be utilized in this.

For information sources google and intranet is mentioned as primary sources. Books as sources are regarded as not so usable, as information in books is quickly outdated in technical field. For major incidents there are named contact persons who participate the problem solving in a meeting type arrangement.

When asked about information requests the manager (SSDM1) is named to be the primary source of questions, named manager probably often being a relay between the actual requestor. Business users (customers) are assumed to take service on "as-is" –basis, not being interested in how the service is run.

Decisions made by SM1 are described mostly as technical, regarding daily service running activities: When updates are installed, when the maintenance breaks are scheduled etc. Internal service is regarded by SM1 as being slightly more difficult in this sense, as all breaks have to be communicated and coordinated between internal customers, who tend to be more demanding in this sense and try to make their own suggestions as how and when the service maintenance should be done.

---

[6] The services discussed in this paper are internal services in a large company in electronic devices business. External companies using these service may be owned by the company running the service, or they may be collaborators.

[7] The Lightweight Directory Access Protocol (LDAP) is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network.

When asked what should be monitored the amount of service requests (count) is mentioned, as it tells about how much resource is needed for the task. Based on the usage the resources can be increased or even decreased if needed. Another item that should be monitored is time required for processing the service request, or total time for a service request to go through the service process. It is mentioned that monitoring requests should be divided by platform (operating system), as what is regarded as "normal" operation parameters differ between platforms and some deviations may be left unnoticed if only average values are monitored. Also it is mentioned that there are different use cases in usage of the service, which are different in what is regarded as "normal". Also automatic and manual processing should be handled separately in order to get more accurate results.

The main driver for monitoring processing time for a request is to anticipate resource usage and depletion, in other words, resource optimization.

It is noted that for outsourced teams the sole contact point for business (customers) is the Solution Delivery Manager, who then is in contact with service manager and other parties. In internal team this is not always the case.

For frequency of the reporting SM1 would prefer online tool that has up-to-date information and that can be queried when needed. Monthly reporting is on the other hand seen as adequate tool for monitoring long term trends in service usage and resourcing.

One specific monitoring item was identified: If service request processing time rise over some threshold even in single event it should trigger an alert, even if the average service request processing time would be within acceptable limits. This way the problem could be noticed much faster and the problem solved.

It is mentioned that false automatic alarms are harmful, as they may lead person to ignore alarms even if they are not false. In one case mentioned in the interview it may have been that something changed in the system, while the monitoring system was not altered accordingly, leading to false alarms being sent.

For SM1 it would be preferred to get reports or data on a daily or weekly basis, but only when needed. Also it is mentioned that the data should be divided by use case, category or product, instead of showing the gross count of service requests. The actual action or decisions which

would be based on this information is left somewhat unclear, only example being removal of unused categories for security reasons. For presentation of the data, charts or trend lines are preferred.

Some frustration about lack of technical, system administrator type of data is expressed, while somewhat outside of the scope of this interview and thesis, as it is know that the source data does not include state of hardware type of information.

3.5.3 Director of Product Lifecycle Management (DIR)

The Director (DIR) interviewed told being responsible manager for 11 areas, Signing being one of them. The director DIR noted that Signing team differs from other areas being service focused, while other areas mainly focus on product development tool management. DIR mentioned that the Signing team is easier to manage, as suggestions for decisions are readily prepared by the SSDM1 and requiring less knowledge of technical details.

When asked about decision initiators DIR mentioned that approximately 10% of decisions have roots in company or department vision or strategy, 30% business unit needs (customer needs), and 60% are initiated by the team or SSDM1, and in general over half of the decisions are initiated by subordinates.

Monitoring reports by DIR is done on business-as-usual basis, with traffic light type of presentation. Green light means no need for action, while yellow and red require further analysis and decisions. It is mentioned that one yellow does not necessarily require any action by the director, but rather a longer lasting trend indicated by the yellow traffic light. In monthly reporting cycle that would mean three month monitoring before action. Importance of problem awareness is emphasized by DIR. It is implied that while monthly reporting is fine with business-as-usual approach when no immediate problems are experienced, but problem situations should preferably be reported and communicated when encountered, major incidents in days or even hours.

Monthly reporting with KPIs is mentioned rather being a collegial tool for communication between managers under the DIR than a tool for reporting to the DIR. It is said that the DIR does not necessarily have time for studying the details, but that he would occasionally be interested in seeing the logic and numbers behind the report. Trend line type presentations is

seen as an important tool in following development of the service and to notice deviations in normal operation mode. A specific example is given: following number of service requests per product program, requiring further investigation if one product program is having double the amount requests compared to other product programs.

DIR concludes by describing three classes for reporting:

1) Business as usual with monthly reporting
2) Short-term escalation – reacting to problems and communicating them
3) Following numerical trends in proactive manner

Some of the trends that would be followed by the DIR if offered the possibility to do so named:

- (Software) build times
- Amount of builds
- Source code database size, or disk usage by product development
- Amount of service requests

Trends like these have been monitored in the past and were used to monitor effects of management operations.

Reports going one level up from DIR have the service statuses as color codes in one slide; service specific data is no more presented at that level. According to the DIR escalations and information how the escalations have been handled should be reported and this information also reported back to the teams.

When asked about information sources Google is mentioned, but questions requiring specific technical knowledge are generally forwarded to subordinates.

About information flow and overflow it is stated that in normal mode the email communication can be handled at the DIR level, but the capacity is easily exceeded in case of longer vacation or such, resulting email questions left unanswered.

3.5.4 Service Manager 2 (SM2)

Service manager responsible for another service in different team was interviewed.

Main challenges mentioned by the SM2 were budgetary, while global reach and wide user base of the service were also mentioned, as well as inexperienced users creating excessive load on the service.

For decisions made SM2 mentions resource management while trying to satisfy customer needs. Organizing daily work and resources is also mentioned. Customer needs are seen as initiating decisions, as well as technical problems in the services that either need to be repaired or avoided in proactive manner.

SM2 sees people as main source for information, while routine work is automated, consuming only about 20-30% of SM2's work time. SM2 mentions being surrounded by experts of various areas, so asking their opinion comes naturally for him.

Main items monitored in the SM2's service are uptime, target being 100% availability of the service at all times, and transfer speed. It is mentioned that data transfer speed is important, as slow transfer speed also slows down software development processes. Unavailability of the service is monitored and alerts sent in case of failure. Essential indicator of the service performance is said to be customer satisfaction. Traffic light approach is also mentioned, green light signaling "everything OK". Alerting is based on preset limits, while trend line type monitoring is more manual, as one has to take in notion if an observed change in curve is due to planned action and therefore normal or a signal of change that requires reaction. In some case change rate is also automatically monitored, so that if growth is faster than anticipated it triggers an alert.

In SM2's opinion there is a limit to how much it is reasonable to monitor, excessive monitoring causing unnecessary load for the system being monitored, and number of false alerts tends to rise. According to SM2 instead of monitoring "everything" the key to unexpected incidents is to have competent people tackling the problems when they are noticed. A competent maintenance person is assumed to be able to repair all but the most severe problems.

It is also emphasized that only relevant information should be reported, so that all reports have a specified function and/or need. Another aspect is reliability, as decisions cannot be based on unreliable data without considerable risk of failure. Reliability of monitoring should therefore be ensured by monitoring the monitoring system.

SM2 concludes that good people are essential to achieving good results, much more important than good information systems.

3.6 Conclusions from the findings

A few general properties for a reporting system can be deducted from the interviews:

1) Reporting and monitoring should have a clear purpose. Every reporting item should have a clear meaning and reason why it is being reported.
2) Data gathered should be reliable and/or results verifiable.
3) The monitoring should not cause any significant load to the system or service being monitored.
4) The information should be up-to-date, preferably available at any time.

Some problems not directly related to the system providing data and reports were identified. First, the "traffic light" type reporting to director is vaguely defined, being essentially based on feelings of the Service Manager and the Senior Solution Delivery Manager. This is not necessarily a problem, if the role of the reporting is seen as a communication tool between the team and the upper management. This works as long as the trust relationship between the management levels is maintained, as problems are expressed only if the manager responsible of them so wishes. Automatic reporting that would effectively nullify this risk would require defining the conditions for green/yellow/red light in a more detailed way. Open reporting and having access to dashboard type information at any time would probably be profitable in the long run even in healthy trust environment as experienced during working on this study.

It was clearly identified in the interviews that monitoring "everything" and reporting about things just because they can be reported is not a preferred way. There should be a clear reason why something is metered and monitored. In talks outside the interview there was an anecdotal remark that "the usual way is that only the things that are metered get done". This however would not mean that all the things that are metered get done, but signals that the *metering combined with goal setting* would gain best results.

Therefore the metering and reporting should be in talks in very early stage of planning, preferably integrated in the process somehow. The information gathering, metering, reporting, should be evaluated regularly. In agile methodologies, like Scrum, periodical retrospective

meetings are as a part of the planning and team work process. Evaluation of metering and reporting could be integrated as a regular part of this process, so all team members could express their opinion if the things being metered are providing correct and useful information.

3.6.1 Identification of areas for monitoring

Two different areas of interest can be defined:

1) Technical environment (servers, network, etc.)
2) Business related (customer satisfaction, recognize main customers, monitor process performance like process duration etc.)

For both technical and business monitoring there are two aspects of monitoring:

1) Monitor development to recognize development trends, to proactively see need for change and planning. Reports and trend lines are tools for this.
2) Monitor the service for **disruptions**. Any change that requires immediate action should be recognized. Alerts are tool for this. To recognize disruptions clear limits for normal operation mode, or business as usual, should be defined.

While not stated explicitly by SSDM1 it can be assumed that creative work is regarded as more complex than the routine maintenance and monitoring of the service, thus leading to enhanced need for information about the state of the service. This is also mentioned by the SM1. It can be seen from expert point of view that any enhancement of the service is a change that results in uncertainty of the status of the system, and there is always a possibility of error, malfunction and suboptimal performance, due to a possible human error in programming or configuration of the service. This can be also seen as last stage of software testing activity, post implementation testing, happening in production environment.

The motivation problem identified by the SSDM1 can be seen as a problem in using human resources in information gathering, a sort of human error in the reporting process. It can be assumed that this kind of behavior is strengthened if the person is under stress and has limited schedule. Automatic gathering of such information could be used to eliminate this problem, or if automation is difficult, the task should be made as effortless as possible. Further interviews should be performed to determine how this could be accomplished.

Monitoring change - or deviations from *business as usual* -situation - can be seen as a central need in all interviews. Trend lines are seen as most useful ways to see if there is a deviation in normal situation, be it static, increasing or decreasing. Trend lines are also preferred as a way to proactively plan improvements required by the trend, for example predicting when more computing power or disk space is required for the service.

Need for differentiation of source data by use case and/or customer is recognized both by SSDM1 and SM1, and also implied by DIR to some extent.

One problem with log source data was recognized by SM1; the lack of user information in logs. In implementation phase it should be studied if this lack of user information should be addressed by altering the logging process, or if the information should be acquired in post processing phase. This is considered being outside the scope of this thesis.

It seems that a service manager in internal team is to some extent handling information requests directly without his manager acting as an intermediary. Therefore it seems reasonable that the same information that is available to SSDM1 should also be available to SM1.

3.6.2 Usages of the monitoring system

Based on the interviews I identified four central usages for the log monitoring system:

1) Monitoring technical aspects and functionality of the service
2) Monitoring costs
3) Monitoring usage and users
4) Reporting to higher level management

It should be noted, that in most interviews the need for information was mainly technical and not directly business decision related. These technical issues included things like use of disk storage space, availability of the service, server load, data transfer speed in the network, and others. I define this type of information as *technical awareness.*

Characteristic for this type of information is that there is a (technical) situation that can be defined as *normal operational mode.* The two aspects of the technical awareness are:

1) Resources
2) Problems

Technical resources monitoring can be seen as a way of verifying that the system running the service has enough resources to fulfill its purpose. Awareness of both current situation and historical trend is seen as necessary in the interviews. For example, monitoring of disk space could be implemented so that the percentage of free disk space in the server is shown, as well as a trend line displaying same percentage for every month over a period of 12 months. It should be noted that it is difficult - if not impossible - to create this kind of information accurately without knowledge of the history of the technical system. For example, if server hard disk space was increased at some point during the period being plotted, it cannot be automatically taken into account when the search engine formulates the trend line.

A problem can be defined as any deviation from normal operation, which in time would result reduced performance and/or unavailability of the service. Problems in service would cause losses to users of the service, as their pending work is delayed, which causes employment costs and may even delay product sales decisions and company revenue from the product delayed. In a company doing business in high technology industry, with relatively fast-paced development cycle and products that have relatively short sales period, any delay in sales decision can result reduced revenue.

Another feature to be monitored, described by SSDM1, is events that indicate cost or expense. A direct cost of the request can also be seen as a variable cost and basis for pricing the service, which makes it reasonable to report these costs by customer, even if the service is internal and being offered without actual direct cost to the customer. In the service being studied the result of a service request event in the service is a certificate, which has a maintenance fee. This maintenance fee depends on number of active certificates[8], which should therefore be monitored. This can be defined as cost awareness.

It should be noted that in this context the cost is a direct result of a user submitting a service request and not any cost resulting from running the service, e.g. computing costs or data transfer costs. A certificate and the cost for it only exist if there is a signing request submitted that

[8] Certificate is valid if not expired or revoked. Certificates have limited validity period and they may be revoked during that period for security reasons.

results such a certificate creation. Running the service can be seen as fixed cost, whereas the certificate cost is a variable cost.

Common for all interviews was the interest for *trend lines,* monitoring for changes in the service. This was common both for technical issues and business. There was strong consensus that a trend line based on monthly (average) values and spanning a few months or maybe a year back would be sufficient. SM1 described that the actual subject of interest is the angle of the trend line. This can be seen as proactive tool to foresee if and when the trend, will render current assumptions of the normal operational mode obsolete, and a change in service or its resourcing need to be considered to ensure service continuity.

We could define the usage of the trend lines as *change awareness,* but this term would not suit well to the same level as technical and cost awareness.

For a trend line two actual uses can be seen:

1) Forecast when current trend will result a need for action, and
2) Evaluate if there is a sudden change in the angle of the trend line, due to unforeseen change in service usage, implying that there might be an issue requiring further study.

SM1 made a notion that the features to monitor in a service vary by the *use case* for the service. That is, an event may have attributes that in one use case fit inside parameters of normal operation mode, while similar event with similar attributes in another use case might be an indication of a problem.

It was also noted in interview of SSDM1 that separate use cases - or separate customers – and their operations are interesting. For example counting number of service requests of one customer offers some information about how that particular customer is using the system. It also provides the manager (in this case, SSDM1) information about how things are in the company, which organizational units are most active, which products are approaching sales decision, and so forth. This information is not necessarily required for making decisions but offer information the manager can use while acting as an information hub (see Mintzberg, 1990).

In light of the interviews, monitoring the service targeted for an internal customer is based on at least these goals:

- Monitor costs resulting from service running
- Optimizing cost of service running by avoiding excessive usage of resources
- Minimize cost (time or money) for the customer
- Ensure continuous availability of the service by identifying problems and development needs

3.7 Decision making in light of the interviews

It was mentioned by the SSDM2 that no actual decisions are made by the managers anymore and that accountability has been reduced lately. In this light it can even be argued that decisions are made elsewhere and not by interviewees of this study. Therefore in this study proposing decisions and/or solutions to problems is regarded as decisions. This principle is identical to that of Byström's. It is obvious that the actual decision itself is made based on these proposals, but the information from the service is required for making the proposal. It is outside of the scope of this study - and candidate to further research - to study what information affects the actual decision, or acceptance of the proposals.

It was identified in multiple interviews, especially in SSDM1 interview that the knowledge of system status is clear input for a decision, the base decision being not to react at all situation being normal, "business as usual". Decision for action is usually initiated by change. Monitoring the rate of change, identifying sudden and radical change, and verifying that changes initiated by need and implemented in the system do not have negative impact in the service is in key role.

3.7.1 Decision process analysis

In making decisions related to the services, we can identify three levels of interest. In Figure 1 I have collected some points of interest for different levels.

In the lowest level, we get information about what I call *events.* An event is when a Service Request is created, a service request is fulfilled, an error happens during the handling of service request, and information related to these.

On higher level we are interested in Service Requests in bulk; for example what happens related to one customer, or related to one use case (an aspect of service, such as signing software packages cryptographically with a certain key).

On even higher level, we are interested in how the hosting environment works, if the service has enough resources to handle the Service Requests received, and the budget.

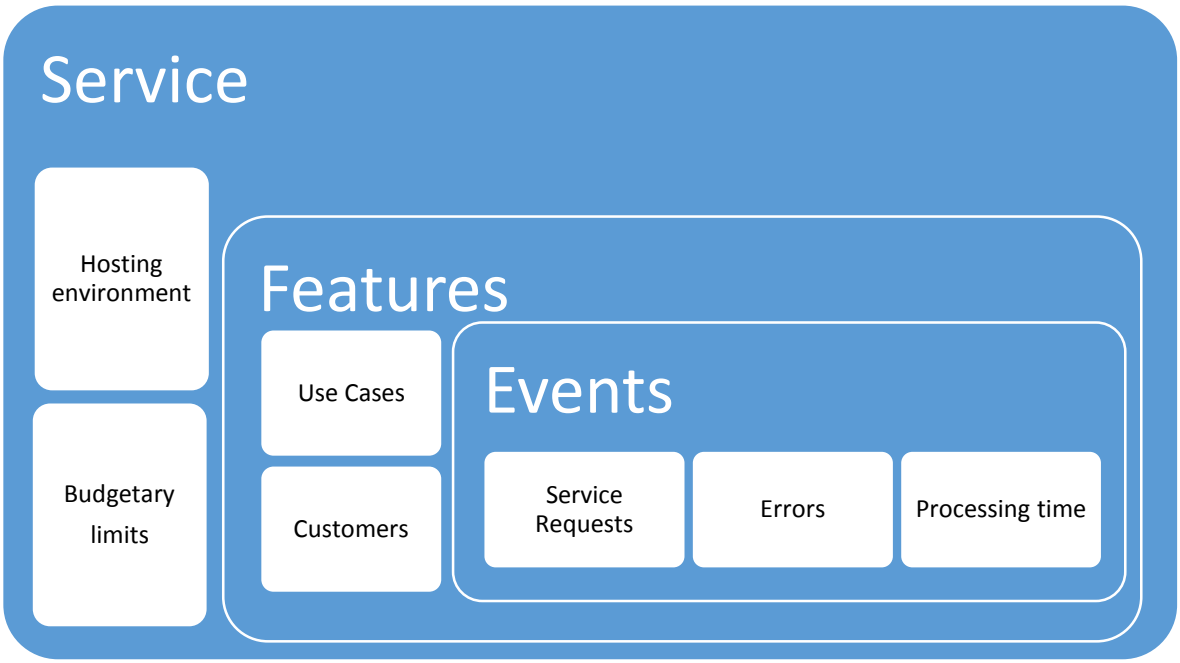


Figure 1: Service log data model of interest

It should be noted that in higher level it is often necessary to get information from the lower levels, but rarely vice versa. Someone making decisions based on error messages is not usually concerned about budgetary limits, or if they bump to such limits while working, usually have to escalate the problem to someone with more decision power, like a manager.

What can't be seen in Figure 1 is that there are also aspects of the service being monitored, the technical, and the (business) process. Typically monitoring the technical system status translates to reports while in the normal operation mode, and alerts when normal operation mode is disrupted. These result different decision situations.

An alert requires immediate action. Usually the situation described by the alert is verified, and if there is an error, corrective actions are performed. The decision is therefore two level: First, decide if action is required. Second, decide what to do to return to normal operation mode. Both of these in low level require technical expertise from the person investigating the

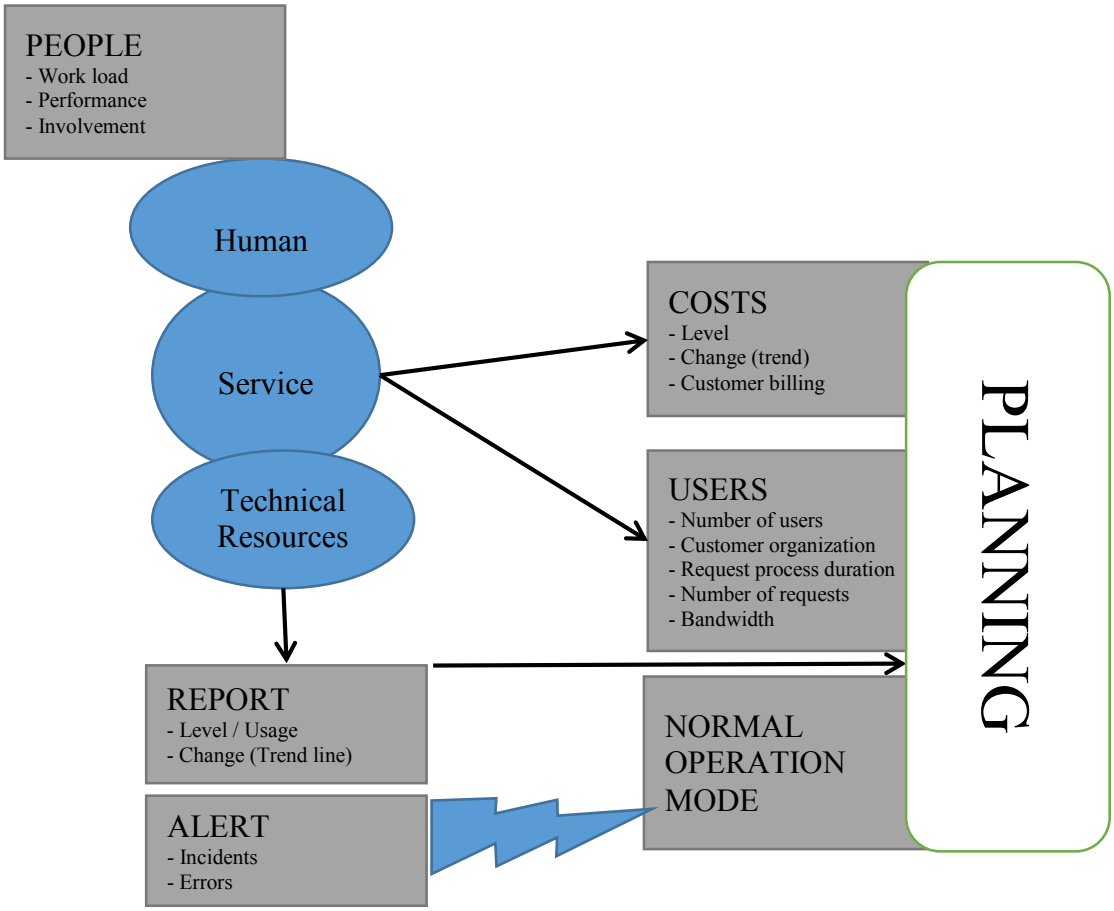
situation; or as SM2 pointed out in the interview, a competent maintenance person is assumed to be able to repair all but the most severe problems.

Ideas about how one should react to Reports vary somewhat. Based on the interviews, especially SSDM interview, if the report indicates that the service runs within acceptable parameters, or "Business As Usual", no action is required. In this sense, to do nothing is also a decision. One should note however that doing nothing when you do not know what to do is not same as making a decision of doing nothing. Reading a report and not understanding what is going on and then ignoring the report is not a real decision in this sense. For this reason, the parameters defining Business As Usual should be defined, as they often are in form of Key Performance Indicator (KPI). If the report indicates that the service is *not* running within parameters, it indicates there is a problem that requires urgent change.

Third possibility is that a trend line type of report view displays that a trend line is developing in a way that at some point require change, like increasing the computing power or storage. In this case, some kind of planning activity should be started as reaction to the emerging need proactively, before the situation develops into a problem requiring urgent action.

In Figure 2, I have summarized the monitoring activities and aspects of the service, related to the planning activities. The human resources are not usually part of the service planning, at least not in context of this study, although workloads of individuals involved in running the service need to be taken into account during planning process.

Figure 2: Service monitoring and planning



4 IMPLEMENTATION

4.1 About Splunk

Splunk Enterprise is a commercial tool for indexing, searching and analyzing the log files. It can handle many kinds of log data, for example Windows Event Logs and traditional text based log files. While it is a commercial software and relatively expensive, there is a free license which offers the same indexing and search functionality as the commercial license, with some restrictions.

Splunk search is a typical Boolean system, where the result set can be limited by defining search terms. Wildcards can be used. Statistical operations can be performed on the result set, but the search engine itself is not probabilistic. Normally query terms are handled like with Boolean AND operator, but operators like NOT[9] can be used to change that behavior.

In this implementation I will focus on data sources, defining search queries, and creating visualizations of the search results. Features of Splunk such as automatic PDF report creation, optimizing the performance of indexing or search, or detailed technical aspects are not in the scope of this study. The search queries are included in Appendix 1.

The version of Splunk used was 6.0.2. build 196940. New versions of Splunk are released quite frequently, so some of the features or issues reported may not apply to other versions of Splunk.

4.2 Information needs identified

The following information needs were identified:

1) Problem management. If the Normal Operation Mode is interrupted for some reason. Result for this is an alert, preferably with a message containing necessary

[9] The Boolean operator should be capitalized to be recognized as operator by Splunk search.

information for verification of the error situation. These are treated as requiring immediate action.

2) Identifying important customers
3) Identifying customers having most problems or worst performances
4) Long term trends and development by service / use case / customer

4.3 Source data evaluation and formulation

As the log data is static and information can't be easily added to log rows afterwards one should be very careful what to log and what not. While it is possible to add logging points and create more than one row for one event for additional information it is wise to carefully plan what to log and how.

If we think of the log file as a database, a data row in a log file can be seen as *a tuple*. In the simplest form the tuple consists of a timestamp and a message, but there is no limit to the complexity of the data row. It is common to use ordered list of values, or key-value –pairs[10]. In this approach the log file forms *a relation* in a relational database model. Therefore with properly formulated log files and suitable database engine or indexer one can apply relational algebra when formulating queries. This is also supported by the Splunk search engine by operators like AND and NOT.

Basic approaches when creating logs are:

1) creating log rows with as much information about the event as possible in one row
2) creating log rows with single bits of information

In option 2 analyzing several rows are needed to gather same information as analyzing one row in option 1. The second option is common if the log is used only to signal about errors and provide debugging info in "timestamp & message" manner.

[10] For example: Name="John Doe"

For determining the duration of the process, full or partial, a log row should be written for every step or checkpoint of the process that is monitored. This comes somewhat naturally if any change of state in the process is logged. For some process without many phases, it might be enough to log just the start and end of the process. It may even help to carry process start timestamp in subsequent log messages related to the service request in question.

It is common that the log messages are created – if they are created – only thinking about detecting problems, errors and failures at hand. For log analysis, it would be beneficial that the structure of the log be such that it would be easy to analyze the log automatically with proper tools. The caveat here is that programmer only communicates the states of the *system* via the log, and not the process or the service offered. For simplicity, it would be reasonable for these two to exist separately and independently in different files. They can, however, be combined in one file if the rows are identified adequately, so that irrelevant rows can easily be filtered out of the search results. This requires some processing time, which might be an issue in a system under stress.

While implementing the system used in this study it was clear from the beginning that the actual log did not include information needed for analysis. Therefore the data was collected to a "pseudo-log", by producing a log-like stream of data rows from the main (SQL) database of the system. The advantages of this over querying the data straight from the database is that the pseudo-log does not depend on the database engine, the pseudo-log is permanent and does not change, and changes in the database do not change the data in pseudo-log. In this study access to the database server was also limited in a way that prevented querying the database from the Splunk server, thus requiring an intermediate providing the data; more about this in section 4.3.2.

4.3.1 What to include in the log

For normal service process we can identify several points of the service process that would provide information when logged:

1) The initiation of the service process (service request). This can be a submission by a customer or any other point-of-entry for information regarding the service request. Usually at this point the request is assigned with an identification code of some kind, which can be used later to analyze flow of the request in the process.

2) State changes for the request. Also here the ID of the request should be visible.
3) Any failures in handling the request.
4) Success in providing the service requested.

What information the log messages should include should be determined separately, depending on the information need. It can, however, well be that the technical implementation sets limits to the information that can be written to the log. There are also other limiting factors that should be considered, for example legal questions concerning privacy of the persons using the system.

4.3.2 Data sources in this study

In this case study, three kinds of data sources were used: Log files of an older system, log based on information stored in an SQL database, and log files from a new system developed in the company.

The first data source consisted of actual log files of an information system tailored for providing content signing service. I will call this data *Source Data 1*. The system had been in use for several years, and the logs produced by the system were not tailored in any way for search and analysis or reporting purposes in mind.

Result of all this is that the information in the logs is somewhat vague and poorly formulated and lacking structure, making automatic processing of information difficult at times. As the system was not under active development at the time, it was not possible to alter the logging principles, and therefore it was soon realized that this data source was not good enough to build much more than a traditional "search the logs to find diagnostic info in case of problems" kind of a system. It was clear from the beginning that another data source was required to get any useful results from this system.

All the information needed for reporting was known to exist in the main database of the service, but the information was scattered across several database tables. Additionally, access to the database was limited due to security reasons. While there is normally the option to configure SQL database query input for Splunk, it was not possible in this case. To gather required data from the Oracle SQL database the main database of the system was polled with a custom made

Java Servlet[11], which was then queried over HTTP connection with a Python script, scheduled to run frequently in Splunk®[12]. The Python script creates a file, which essentially forms a CSV-formatted log file, which is then indexed in the Splunk engine to facilitate searching and analysis. The script reads the last timestamp in the file and uses the timestamp as parameter when querying the Servlet, which then returns all the events, registered to system after the time stamp. I will call this data *Source data 2 (SD2)*. While there were two files created for two use cases, these both have quite a similar syntax and data and they are in this study treated as one source data type.

To list SD2 service requests from the database an SQL query was formulated. As the information system was old and the database is somewhat complex, the resulting query was impressive in complexity, and not very intuitive. See Figure 3: SQL query to produce log rows from Oracle DB. Fortunately, formulating the SQL query is a one-time effort.

The program code for the Python script as well for the Servlet are included in the Appendices 2 and 3, respectively. As the writer is not a professional programmer the examples should be treated as such; there may be bugs in the code, and the structure of the programs may not be optimal. Some shortcomings were identified during several months of gathering data to logs. More of this in chapter "Evaluation of the implementation".

[11] This somewhat complex arrangement was needed, as access to the data and the database was limited due to security reasons (firewalls, server administration etc.).

[12] Splunk® Enterprise version has a built-in Python 2.7.5 interpreter.

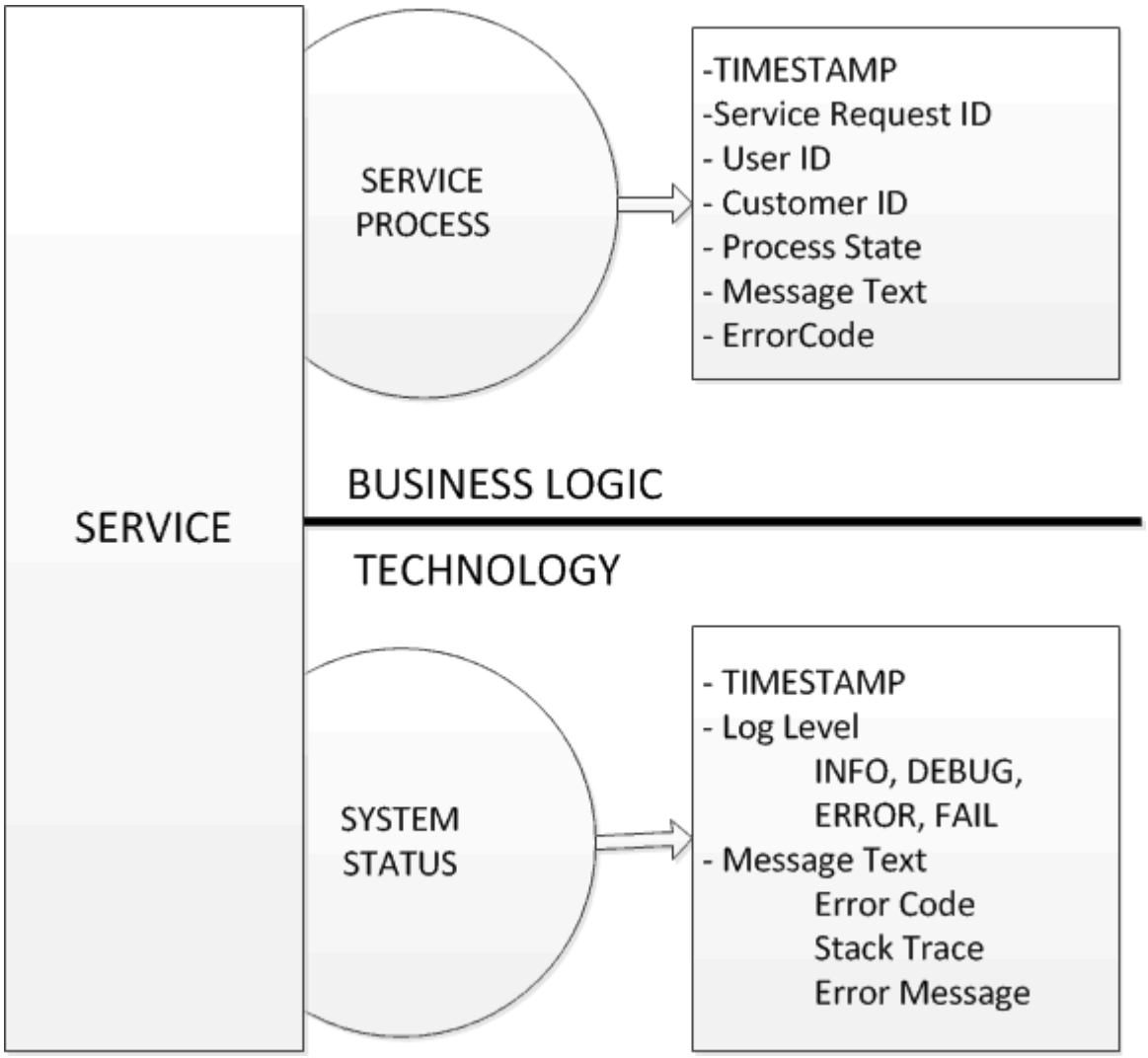


Figure 3 Two aspects of Log Formulation

The third data source was the logs of the newer service being under continuous development by the team, so the data written to the log file could be influenced and altered. As the logging was only a minor feature of the system only few changes were requested after the implementation, mainly after some bug or unexpected data was observed. I shall call this data Source Data 3 (SD3).

Table 1: Source data types

Name	Type	Source
SD1	Application Log (Log4j)	Legacy Application
SD2	CSV Pseudo log	SQL Database Poll
SD3	Application Log (Log4Net)	Application

```
SELECT * FROM (SELECT ra.action_time AS ACTION_TIME, ra.action_type AS ACTION_TYPE, ra.user_id AS
ACTOR_ID, des.description AS STATUS, req.request_id as REQUEST_ID, req.owner_id AS DEVELOPER_ID,
req.owner_name AS DEVELOPER_NAME, req.source AS IS_EXTERNAL, wf.workflow_name AS WORKFLOW,
req.request_date AS SUBMIT_DATE, cg.name AS CATEGORY, type.name AS PLATFORM, nu.username AS
ACTOR_NAME, (SELECT ROUND(SUM(rf.file_size)/1024/1024,3) FROM service1.request_file rf WHERE
rf.request_id=req.request_id ) AS FILES_SIZE_MB FROM service1.request_action ra INNER JOIN
service1.request req ON req.request_id=ra.request_id INNER JOIN service1.stattool_request_state_desc
des ON req.state=des.state INNER JOIN service1.category_group cg ON req.cg_id=cg.cg_id INNER JOIN
service1.type type ON type.type_id=cg.type_id LEFT OUTER JOIN service1.workflow wf ON
req.workflow_id=wf.workflow_id LEFT OUTER JOIN service1.service1_user nu ON
ra.user_id=nu.employee_id WHERE ra.action_time > to_timestamp('"+dateString+"','YYYY-MM-DD
HH24:MI:SS.FF3') AND ra.action_time <= sysdate ORDER BY ra.action_time ASC) WHERE ROWNUM < 10000
```

Figure 4: SQL query to produce Source Data 2 log rows from Oracle DB. The `"+dateString+"` embeds date parameter to the query in Java. Number of results is restricted to 10000, as Tomcat Catalina Servlet engine was prone to crashes with large datasets.

4.4 Alerts

Alerting is the basic tool when there is a sudden malfunction or other disturbance that requires immediate attention from the administration of the service. Candidate events for alerting are *technical failures* (such as disk full -situations), *process failures* (such as request failing unexpectedly) or *security problems.* In this study, security problems are considered being out of scope, as they are very rare in internal services, where customer base and access to the service are limited. In a service that is run open to the internet security problems would require considerably more attention.

4.5 Trend lines

The implementation of the system with Splunk had four phases:

1) Determine if the source data has enough information so that required information can be searched from the data. Define which information entities are needed to formulate information and knowledge of the data
2) Create a search to create result data set matching the information need
3) Define analysis methods and post-processing operations for the data sets, to refine results to provide needed information or knowledge
4) Select appropriate visualization to facilitate understanding the results

4.6 Search formulation

When input data containing the required data is identified, we are ready to formulate a search to refine the source data to useful result set. For this we need to know several things:

- What is the core, identifying aspect that is measured?
- What unit or scale is used for metering (count, average, maximum)
- Are there some other aspects that should be combined with the identifying aspect

4.6.1 KPI 1: Service Request count

The first Key Performance Indicator for the service is the count of service request. This is one of the key KPIs, as this can be with some accuracy counted both from Source Data 1 (SD1) and Source Data 2 (SD2). Therefore this count can be used to verify that both the data sources are up-to-date, and a large difference between the results can be seen as an indicator of error in (either) source data.

It was noticed that producing reliable results from either data source was difficult. In SD1 there were some difficulties in recognizing the log rows that would identify a request submission. In SD2 the problem was that in the data there were log rows for all kinds of status changes, so that one request could span over two months. For example if the request was submitted on the last day of month, but was processed next month the same REQUEST_ID could be found in both months. It was also unclear whether all requests shared a common entry point, for example if search for “ACTION_TYPE=1” - assumed to mark request entry to the system - matched for all REQUEST_IDs. This can be attributed to the fact that the internal logic of the application was not accurately known, and studying the program code of the system would have taken too much time and effort.

Here are two examples of Search Queries in Splunk for the two source data types. The first query searches for arbitrary strings to identify relevant log rows, while in the second one automatically assigned key-value pairs in Splunk are used:

SD1 (indexed to default index):

```
source="*service1*" "checkuseraccess" AND "\"Standard application\" - start" earliest= 12month@month
| convert timeformat="%Y-%m" ctime(_time) AS Month
| chart count by Month
| fields - _time
| fields + Month, *
```

SD2 (indexed to ”reporting” index):

```
index=reporting  source="request_file_path" ACTION_TYPE=1
| dedup REQUEST_ID
| convert timeformat="%Y-%m" ctime(_time) AS Month
| chart count by Month where top100
```

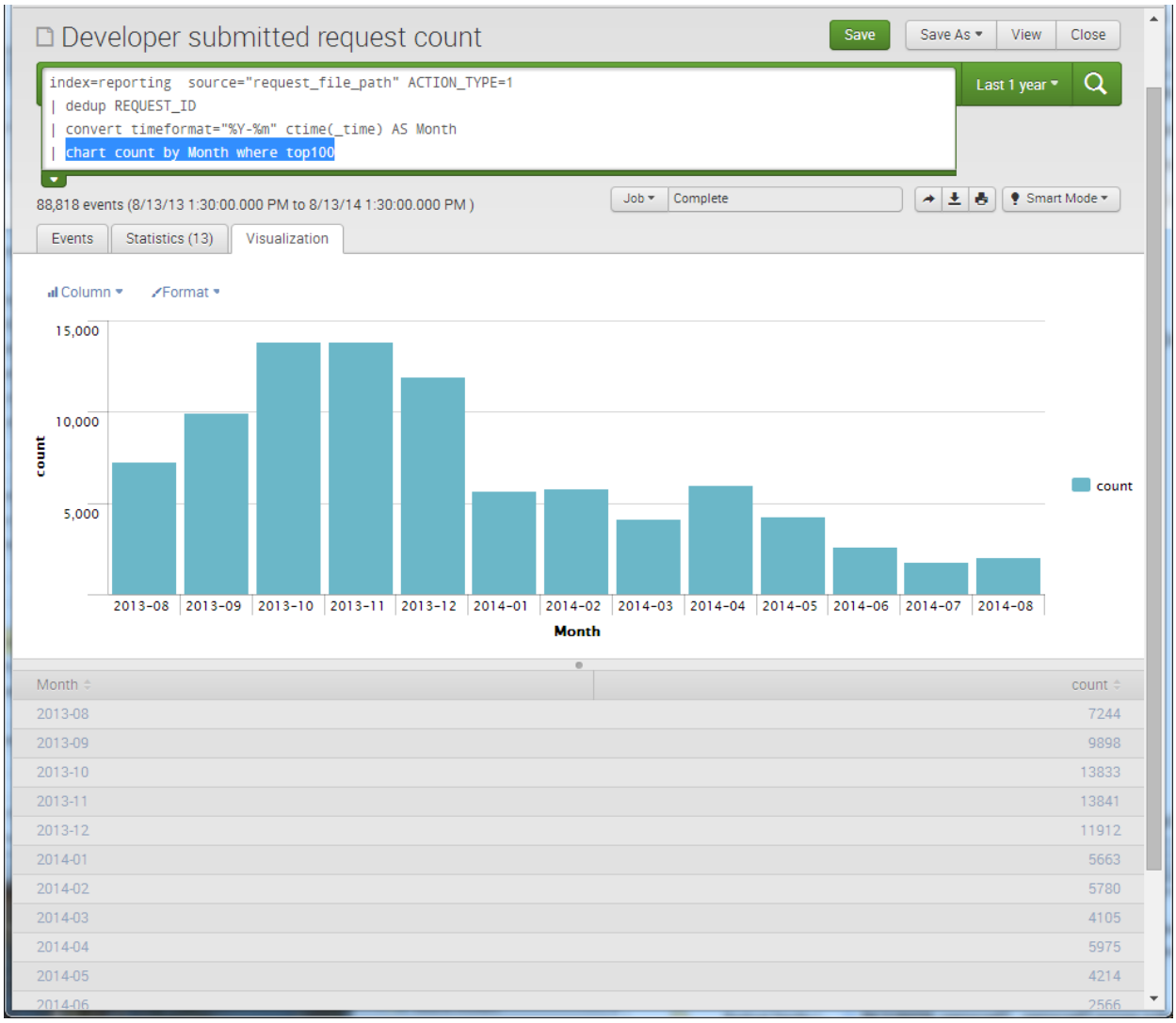


Figure 5: Typical results of a search; here from Source Data 2

4.6.2 Service Manager: Volume of requests in time frame

In one case the Service Manager needed an answer to a simple question: What is the maximum number of service requests submitted in a period of an hour during the year? This information was needed for test design, so that a server load roughly the same as in the production environment could be produced during testing.

This query was first expressed as an SQL query:

```
SELECT CAST(StartTime as date) AS Date,
       CAST(DATEPART(Hour, StartTime) as varchar)+ ':00' as 'Time',
       COUNT(*) AS Totals
FROM [SigningService].[dbo].[Submissions]
where [StartTime] between '2014-01-01 00:00:00.000' and '2014-08-31 23:59:59.999'
GROUP BY CAST(StartTime as date),DATEPART(hour,StartTime)
       order by Totals desc
```

Results of this can be seen in Figure 5.

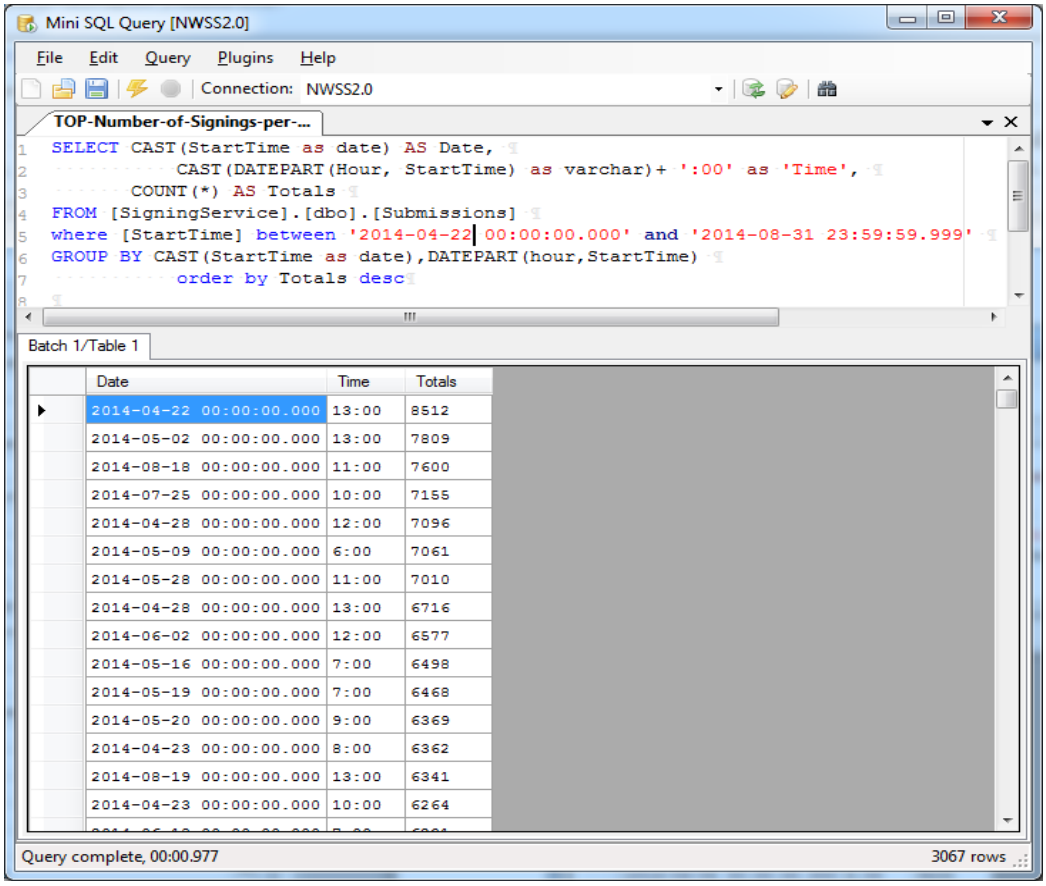
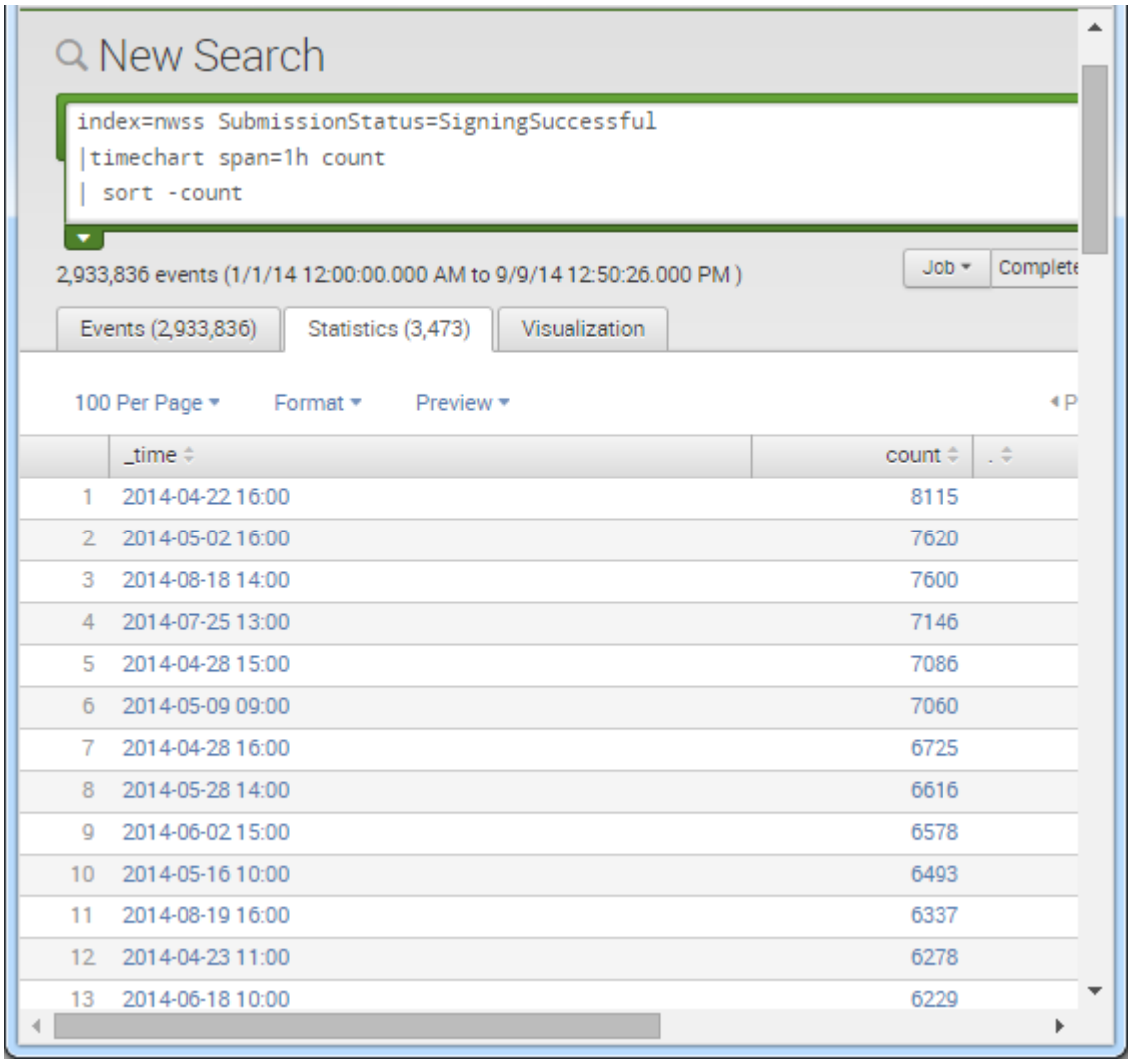


Figure 6: SQL query results

The same data can be produced in Splunk with query like this:

```
index=foobar SubmissionStatus=SigningSuccessful|timechart span=1h count | sort -count
```



Query explained: the index definition specifies the index being used; the different data sources are indexed in separate indexes to help targeting the query to the source data preferred.

`SubmissionStatus=SigningSuccessful` is used for deduplication, as every service request gets two log entries in this setup; at the submission, and at the completion of the request. Timechart is a handy tool in Splunk for analyzing data against time, and `sort -count` specifies that the highest values should be shown at the top of the results.

A few observations can be made from the results. First, the time difference: the Oracle SQL database queried returns the results as UTC, while Splunk timestamps are in the local EET time zone, therefore timestamped three hours later. This is a simple technical feature having no real effect.

Second, and more important: there is a maximum of 4.7% difference in results. The source for this difference is unknown, but if we compare the first 100 results (see Appendix 4: Difference in Splunk and SQL results) of these searches we can see a difference, averaging 1.2 % between the results. While the source of this error is unknown, all but one case in top 100 results the number is bigger in SQL results. Most probably some of the requests either do not get their data written to the log, either due to programming error in the service application, or due to log rows getting lost in transmission between the application and the log server.

After some research a software error (or more like 'a neglected category of requests') was found in the logging interface in the application, leaving a group of requests unlogged. These requests were visible when queried from DB, but not logged in the files indexed by Splunk. This emphasizes the importance of data verification, both when feeding the data to Splunk, and later when using the data for analysis.

4.7 Visualization of the results

To facilitate understanding of the information refined by the search selecting an appropriate visualization for the query is important. Various graphical presentations[13] are available for statistical functions, or a table for pure numerical representation. Graphical presentation is more appropriate for information that is "glanced" without or before a more detailed inspection. For example, a trend line displays how situation has developed historically in light of the data available.

[13] For me importance of selecting correct graphical visualization was emphasized in mandatory B.Sc. module "Introduction to statistics".

5 EVALUATION OF THE IMPLEMENTATION

5.1 Problems in data input

Two major problems were noticed during the implementation. First, the information in the application logs was not well suited for searching, and therefore formulating useful searches for the data was difficult.

Another problem was with the data fetched from the database through Servlet polled with the Python script. While the program worked fine most of the time, sometimes the data rows in the file would end up corrupted. Sometimes a line was split into two, having an extra linefeed in middle of the row. Sometimes parts of the line went missing, leaving some data out. This is illustrated in Figure 4 and Figure 5. In three month period 17 such events were detected. It was peculiar that while the same Python script/Servlet combination was used to query data for the whole year (and even earlier data), these errors were seen only in log rows for last five months of the period.

The problem was that while the rows were incomplete, Splunk would index them like any other row, resulting in peculiar and obviously erroneous data in dashboards. This is especially problematic in Splunk, where correcting indexed data is difficult, if not nearly impossible. In this case manipulating Splunk data was even more difficult, as there was no easy command shell access to the server hosting the Splunk data.

To find the root cause for this problem more testing would be needed, but it is assumed that the problems were caused by HTTP connection, which is somewhat sloppy and lacking data verification. [14]

[14] I was never able to verify the actual source for this error, although implementing verification routines to program code greatly reduced the amount of errors. Single instances of this type of error were still found months after the actual implementation phase was over. It is possible, but by no means certain that the error was partly caused by the indexing in Splunk itself.

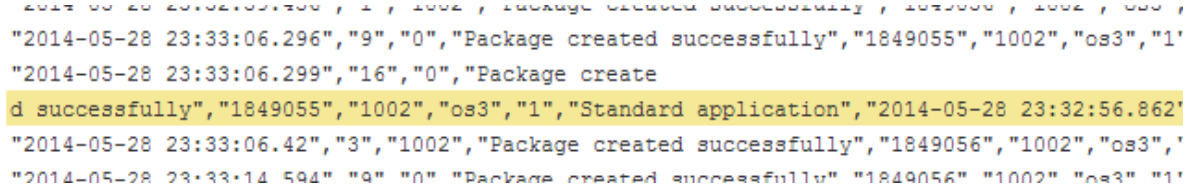


Figure 7: Log row with an extra line feed causing error in index and search. Cause is unknown; possibly bug in either Java servlet or Python script. This emphasizes the need for syntax verification. After indexing this kind of error is very difficult to fix in Splunk; at least the second row without timestamp has to be deleted (or be masked deleted) to prevent confusion. The first partial row, although incomplete, still may have some information value, as Splunk indexes the first three fields correctly.

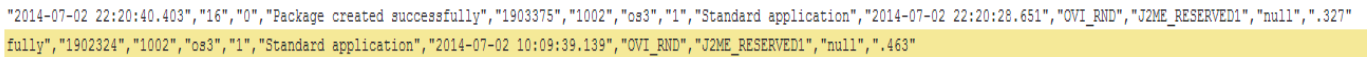


Figure 8: An incomplete log row with beginning of the row missing; note that the previous row is complete. The row is split in the same field as before, a notion which may have some importance in debugging.

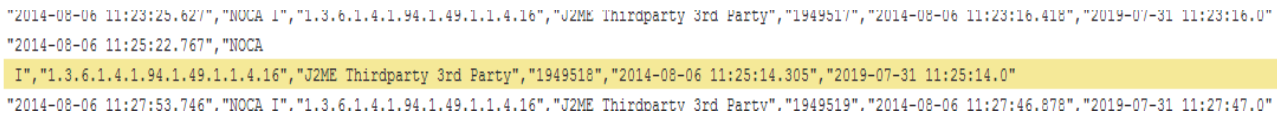


Figure 9: Another split row, from the other Servlet originated (pseudo)log file.

This problem emphasized the *need for verification of the input data*. Adding more thorough verification of the data received from the Servlet into the Python script - as checking that every line received has the correct amount of fields - would probably solve this problem. Any lines received from the Servlet having any trace of errors should always be dropped and not written to the log, script execution should be halted, and another polling attempt be done from the scratch. This is also in line with opinions of SM1 as expressed in the interviews.

The Servlet interface that was required to access the database through the firewall also caused some problems. One thing observed was that the servlet engine (Tomcat Catalina) version used could not handle very large datasets. Fetching several million rows from database first resulted in connection timeouts, and when timeout value was raised, the Tomcat/Catalina Servlet engine stalled, requiring a manual restart of Catalina engine. For this reason size of the result set was limited to 10000 (ten thousands) in SQL query. Running the script frequently enough gradually

built the log file 10000 rows at the time. Frequency of the script execution in this setup depends on the volume of events in the system, as setting too low a frequency would result a situation where the number of events is constantly growing faster than the script can fetch from the database. On the other hand running two instances of the script writing to same file might result in garbled output or duplicate rows, as no locking mechanism for the pseudo-log file is implemented. In a more delicate object-oriented implementation, a Singleton-pattern[15] could be used to prevent such situations.

5.1.1 String validation in Python

As suggested before, there should be a validation mechanism to verify that the event string syntax matches the expectation. A relatively simple Regular expression rule was used. As the file format in this case is CSV (Comma Separated Values) another option would be using existing Python CSV libraries.

```
import re

a = re.compile("^\"([0-9A-Za-z]* \")$")

a.match(string)
```

Figure 10: Simple Regex validation in Python, matching any alphanumeric string enclosed in double quotes.

The Regex described in Figure 9 was formulated to verify line integrity. The idea in this regular expression is that in log lines created by the Servlet always have one or more fields, each enclosed in double parentheses. If there is more than one field, the fields are separated by a comma. Every field in a row can have any other characters except newline, carriage return, and double parenthesis. Each row ends with a newline character code \n[16].

---

[15] Singleton-pattern is a design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. (http://en.wikipedia.org/wiki/Singleton-pattern - fetched 08.05.2015)

[16] Most textual Internet protocols (including HTTP, SMTP, FTP, IRC and many others) mandate the use of ASCII CR+LF ('\r\n', 0x0D 0x0A) on the protocol level, but recommend that tolerant applications recognize lone LF ('\n', 0x0A) as well. (http://en.wikipedia.org/wiki/Newline - fetched 08.05.2015)

```
LINE INTEGRITY CHECK WITH REGEX

^\"[^\r\n\"]*\"(?:\,\"[^\r\n\"]*\")*\n

EXPLANATION:
```

- `^` assert position at start of the string
- `\"` matches the character `"` literally
- `[^\r\n\"]*` **match a single character not present in the list below**
  - `Quantifier *:` Between `zero` and `unlimited` times, as many times as possible
  - `\r` matches a carriage return `(ASCII 13)`
  - `\n` matches a fine-feed (newline) character `(ASCII 10)`
  - `\"` matches the character `"` literally
- `\"` matches the character `"` literally
- `(?:\,\"[^\r\n\"]*\")*` **Non-capturing group**
  - `Quantifier *:` Between `zero` and `unlimited` times, as many times as possible
  - `\,` matches the character `,` literally
  - `\"` matches the character `"` literally
  - `[^\r\n\"]*` **match a single character not present in the list below**
    - `Quantifier *:` Between `zero` and `unlimited` times, as many times as possible
    - `\r` matches a carriage return `(ASCII 13)`
    - `\n` matches a fine-feed (newline) character `(ASCII 10)`
    - `\"` matches the character `"` literally
  - `\"` matches the character `"` literally
- `\n` matches a fine-feed (newline) character `(ASCII 10)`

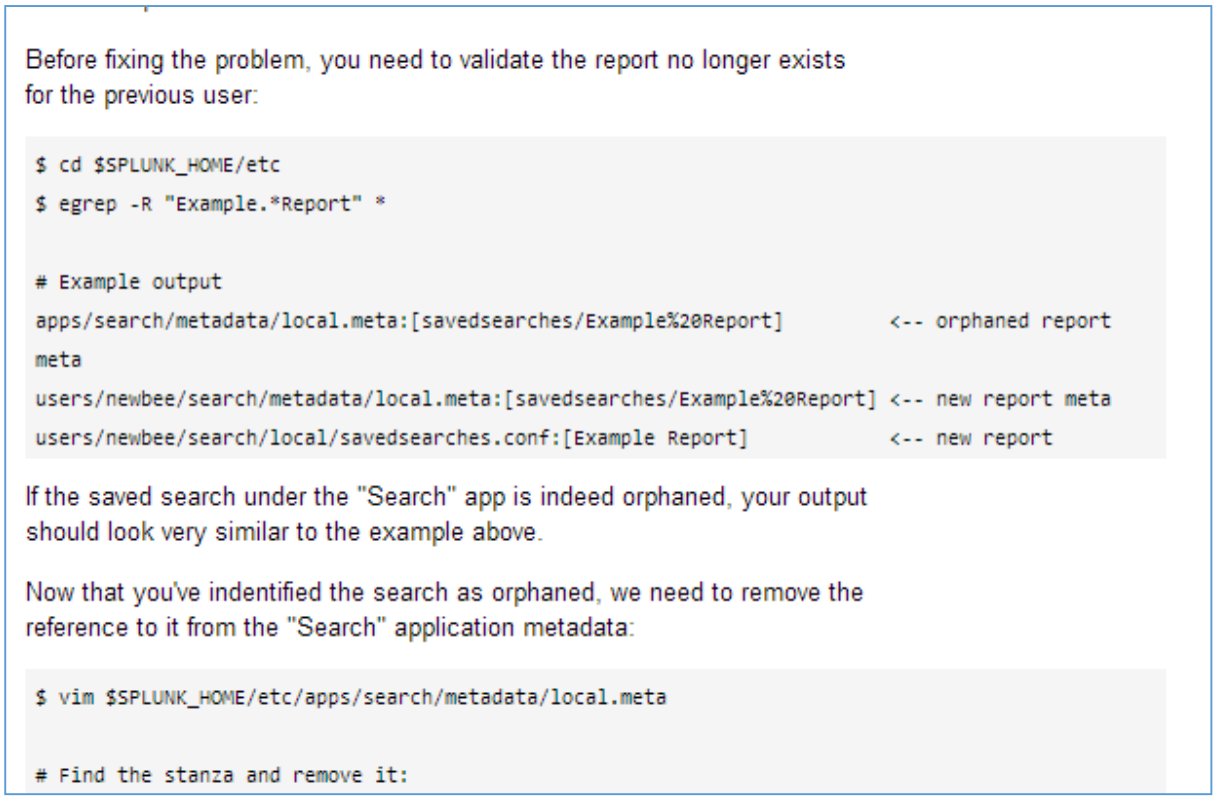
Figure 11: Regex for checking log line integrity. Explanations from site: regex101.com

5.2 Problems with Splunk administration

Some problems in implementation can be classified as originating from Splunk or how the Splunk is designed to be used. The issue that proved most troublesome was that while Splunk has extensive Web-based User Interface, in multiple situations Command Line Interface (CLI)[17] was required to change some setting or fix some error. The problem was that for

[17] Usually CLI - or shell interface - is used with Telnet or SSH client like Putty, which is characteristic for Linux and UNIX server environments.

security reasons there was no shell access to the production system running the Splunk server, but all changes and maintenance required help from maintenance personnel. This made diagnosis and fixing the system very difficult and time consuming.



Before fixing the problem, you need to validate the report no longer exists for the previous user:

```
$ cd $SPLUNK_HOME/etc
$ egrep -R "Example.*Report" *

# Example output
apps/search/metadata/local.meta:[savedsearches/Example%20Report]         <-- orphaned report
meta
users/newbee/search/metadata/local.meta:[savedsearches/Example%20Report] <-- new report meta
users/newbee/search/local/savedsearches.conf:[Example Report]            <-- new report
```

If the saved search under the "Search" app is indeed orphaned, your output should look very similar to the example above.

Now that you've indentified the search as orphaned, we need to remove the reference to it from the "Search" application metadata:

```
$ vim $SPLUNK_HOME/etc/apps/search/metadata/local.meta

# Find the stanza and remove it:
```

Figure 12: Excerpt of an instruction how to fix a Splunk error in CLI. This could not be done through Web interface.

The problem can be attributed to the fact that the implementation project was a pilot project, so the goals and design ideas for the system changed during the implementation and testing. Normally implementation would be done first in a Development environment, and after a testing phase it would be moved (copied) to a Production environment. As the source data - the log files - were only accessible in the Production server this process model was not followed to full extend. For better results, the source data should be made available in the Development environment, so that the functionality of the system could be tested properly. Evaluating the usability of the system for the end user, the decision maker, would have been difficult without the use of real log data.

6 CONCLUSIONS AND FURTHER RESEARCH

There were multiple technical problems in the system implemented. One finding was that Splunk® does not handle input errors well. Fixing anything after indexing is cumbersome in Splunk, as there are really no tools for altering indexed information. This is, of course, how things should be, if the data is to be used as a trusted record of events, like in security related monitoring. This emphasizes the importance of having reliable data in the first place.

Reliability of the data in the logs was found to be relatively low, as there are no mechanisms for ensuring the integrity of the data. Simple syntax checking is one tool for this, but more advanced verification methods, such as calculating checksums to verify that the transfer has succeeded, would be needed if calculations and reports requiring high precision of the numbers are preferred. Calculating a checksum for each row would be near-trivial a task, but verifying that every row gets delivered would be more difficult. [18]

Data corruption due to transfer errors is one source of error identified empirically in this thesis. For example, losing full or partial log row(s) while transferring the data to the log server via network caused bias to the results. This can in some cases be acceptable, if the magnitude of the error is known, the error is small enough, and the results are consistent over time. Some verification method should be implemented to maintain any confidence on the results. Partial log rows on the other hand garbled the results badly, as Splunk for some reason indexed even the partial rows lacking timestamp.

Based on what this study has found log file based reporting is not very well suited for optimization type of decision making, unless the above-mentioned sources of errors are eliminated. Also it should be noted that if the data can't be verified and there is any risk of inaccuracies, the data should not be used as input for other systems. However log file based reporting is useful in decision situations where exact accuracy is not a requirement, or if the amount of events is large enough that statistical analysis can be done even with some missing event information.

[18] Some examples about reliable file updates, including calculating checksums in Python: http://blog.gocept.com/2013/07/15/reliable-file-updates-with-python/

To maintain any confidence in the results of analysis, at least some data verification methods should be implemented. For example, producing a simple event count from database and comparing it to the number of events in the log index would tell if data loss happens during the process. This would essentially be one way to 'monitor the monitor', as suggested by Service Manager 2 in the interviews.

In the end, the suitability of the log data for each decision maker should be evaluated case by case. In this study there was a degree of uncertainty in the analysis that persisted during the study. If the decision maker requires absolute accuracy and control of the data there is no point in creating log based analysis systems, as the reports and analysis created based on the data would most probably be ignored by the decision maker, who would deem the data unusable. If some degree of uncertainty can be tolerated, or if the degree of error in the analysis can be determined, the system may be useful for some decision makers.

One practical technical finding was that indexing and searching (with Splunk) even a quite simple data source requires a lot of processing power and memory, even in modern server setups. This should not be a surprise to anyone who has done any practical work with indexing engines, but nevertheless needs to be taken into account when working with data sources.

For future research it would be interesting to create a normalized test setup, where different searches, different visualizations and different decision situations could be evaluated in a laboratory environment.

REFERENCES

Ackoff, R. (1967). Management misinformation systems. *Management Science*, *14*(No.4), B147–B156. Retrieved from http://mansci.journal.informs.org/content/14/4/B-147.short

Andrews, J. (2001). A Framework for Log File Analysis. *Citeseer*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.28.2519&rep=rep1&type=pdf

Byström, K. (1999). *Task complexity, information types and information sources : examination of relationships*. Tampere University.

Hätönen, K. (2009). *Data mining for telecommunications network log analysis*. University of Helsinki. Retrieved from http://doria17-kk.lib.helsinki.fi/handle/10024/43397

Jonassen, D. H. (2012). Designing for decision making. *Educational Technology Research and Development*, *60*(2), 341–359. http://doi.org/10.1007/s11423-011-9230-5

Järvelin, K. (1987). Kaksi yksinkertaista jäsennystä tiedon hankinnan tutkimista varten. *Kirjastotiede Ja Informatiikka*, *(6(1):1-34*, 18–24.

Järvinen, P. (2012). *On Research Methods*. Tampere, Finland: Opinpajan kirja.

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, *15*(4), 251–266. http://doi.org/10.1016/0167-9236(94)00041-2

Mintzberg, H. (1990). The manager's job: Folklore and fact. *Harvard Business Review*, (March-April 1990), 163–176. Retrieved from http://books.google.com/books?hl=en&lr=&id=N1A4rnVMRuAC&oi=fnd&pg=PA47&dq=The+Manager%27s+Job:+Folklore+and+Fact&ots=2D77kuPvx5&sig=xuYQgiHu5tx_Pi2W3-lQMTxIe_s

Olson, D. L. (2001). Rationality in Information Systems Support to Decision Making, (1954), 239–248.

Sauter, V. (2011). *Decision Support Systems for business intelligence* (2 edition). Wiley. Retrieved from http://books.google.com/books?hl=en&lr=&id=ncNELpySnM0C&oi=fnd&pg=PR7&dq=Decision+Support+Systems+for+Business+Intelligence&ots=2shKpRXE8h&sig=mViNOWRuy-0kQi0m0noZ4qCSXyg

Simon, H. A. (1960). *The New Science of Management Decision*. New York, USA: Harper Brothers.

APPENDICES

Appendix 1: Splunk search definitions

[Request per Month from logs]
search = source="*service1*" earliest=-12month@month "request" "submitted."| rex field=MessageText "request #(?<req_id>.*) submitted." | dedup req_id | convert timeformat="%Y%m" ctime(_time) AS month | chart count as "Signing requests" by month where top100

[Montly CA usage]
search = index="reporting" source="/splunkdata/reporting_prod_certs" earliest=-12mon@mon | dedup REQUEST_ID, POLICY_OID | convert timeformat="%Y-%m" ctime(_time) as "Year-Month"|chart count over Year-Month by CA where top100 |addtotals fieldname="Total" *CA*

[Monthly Policy Usage for Charting]
search = index="reporting" source="/splunkdata/reporting_prod_certs" earliest=-12mon@mon |dedup REQUEST_ID, POLICY_OID|convert timeformat="%Y%m" ctime(_time) AS month| \
chart count over month by POLICY_NAME where top100 |addtotals label="TOTAL" *J* *Mee* *N* *P* *S*

[Active Certs per CA]
search = index="reporting" source="/splunkdata/reporting_prod_certs" |dedup REQUEST_ID, POLICY_OID | eval expdate=strptime(EXPIRATION_TIME, "%Y-%m-%d %H:%M:%S.%Q") | convert timeformat="%Y%m" ctime(expdate) AS expyearmonth| eval timenow=now() | where expdate > timenow | chart count by CA

[Average from-submit -to-signing -duration per Month]
search = index="reporting" source="/splunkdata/reporting_prod_requests" ACTION_TYPE="16" earliest=-12month@month | dedup REQUEST_ID | convert timeformat="%Y-%m-%d %H:%M:%S.%3Q" mktime(ACTION_TIME) as end_time mktime(SUBMIT_DATE) as start_time| table _time, start_time, end_time | eval duration=end_time-start_time | convert timeformat="%Y%m" ctime(_time) AS month| convert timeformat="%H:%M:%S" ctime(duration) AS cduration | chart avg(duration) by month | convert timeformat="%H:%M:%S" ctime(avg(duration))

[Average submit -to-signing -duration per Month hh:mm:ss.S]
search = index="reporting" source="/splunkdata/reporting_prod_requests" ACTION_TYPE="16" earliest=-12month@month | dedup REQUEST_ID | convert timeformat="%Y-%m-%d %H:%M:%S.%3Q" mktime(ACTION_TIME) as end_time mktime(SUBMIT_DATE) as start_time| table _time, start_time, end_time | eval duration=end_time-start_time | convert timeformat="%Y%m" ctime(_time) AS month| convert timeformat="%H:%M:%S" ctime(duration) AS cduration | chart avg(duration) as duration by month |eval duration=tostring(duration,"duration")

[Average time from-submit-to-approver-acceptance per month]
search = index="reporting" source="/splunkdata/reporting_prod_requests" ACTION_TYPE="18" earliest=-12month@month | dedup REQUEST_ID | convert timeformat="%Y-%m-%d %H:%M:%S.%3Q" mktime(ACTION_TIME) as end_time mktime(SUBMIT_DATE) as start_time| table _time, start_time, end_time | eval duration=end_time-start_time | convert timeformat="%Y%m" ctime(_time) AS month| chart avg(duration) by month | convert timeformat="%H:%M:%S" ctime(avg(duration))

[Duration from MngrApproval to signing]
search = index="reporting" source="/splunkdata/reporting_prod_requests" earliest=-12month@month (ACTION_TYPE=3 OR ACTION_TYPE=16)|transaction REQUEST_ID startswith="ACTION_TYPE=3 " endswith="ACTION_TYPE=16" | convert timeformat="%Y%m" ctime(_time) AS month | chart avg(duration) as duration, count by month | eval duration=tostring(duration,"duration")

[Montly Policy Usage]
search = index="reporting" source="/splunkdata/reporting_prod_certs" earliest=-12mon@mon |dedup REQUEST_ID, POLICY_OID|convert timeformat="%Y%m" ctime(_time) AS month| \
chart count over month by POLICY_NAME where top100 |addcoltotals labelfield=month label="TOTAL" *Java* *Mee* *Symbian* *J2ME* *policy*

[Platform-Category Count by Month]
search = index="reporting" source="/splunkdata/reporting_prod_requests" ACTION_TYPE="16"|dedup REQUEST_ID| eval requtime=strptime(SUBMIT_DATE, "%Y-%m-%d %H:%M:%S.%Q")|convert timeformat="%Y%m" ctime(requtime) AS yearmonth|stats count by yearmonth,PLATFORM, CATEGORY | sort -yearmonth, PLATFORM, COUNT

[Requests per Month from DB]
search = index="reporting" source="/splunkdata/reporting_prod_requests" earliest=-12mon@mon STATUS="Package created successfully" | dedup REQUEST_ID | convert timeformat="%Y%m" ctime(_time) AS month | chart count as "Signing requests" by month where top100 | fields - _time |fields + month, *

[SpecialApp Signings Per Month]
search = index="reporting" source="*requests" CATEGORY="*SpecialApp*" earliest=-6months@month| dedup REQUEST_ID | convert timeformat="%Y%m" ctime(_time) AS YearMonth | stats count as "SpecialApp Signings" by YearMonth | table YearMonth *

[Users List]
search = index="reporting" source="/splunkdata/reporting_prod_requests" NOT ACTOR_ID=0 NOT ACTOR_ID=1200 NOT ACTOR_ID=2300 NOT ACTOR_ID=2400 NOT ACTOR_ID=12005 NOT ACTOR_ID=1002 NOT ACTOR_ID=1001| dedup ACTOR_ID |sort 0,+num(ACTOR_ID)| table ACTOR_ID,ACTOR_NAME

[Average MgrApprove - Signing duration per month]
search = index="reporting" source="/splunkdata/reporting_prod_requests" earliest=-12month@month (ACTION_TYPE=3 OR ACTION_TYPE=16)|transaction REQUEST_ID startswith="ACTION_TYPE=3 " endswith="ACTION_TYPE=16" | convert timeformat="%Y%m" ctime(_time) AS month | chart avg(duration) as duration, count by month | eval duration=tostring(duration,"duration")

[Average MngrApprove-ApproverAcceptance duration per month]
search = index="reporting" source="/splunkdata/reporting_prod_requests" earliest=-6month@month (ACTION_TYPE=3 OR ACTION_TYPE=18)|transaction REQUEST_ID startswith="ACTION_TYPE=3 " endswith="ACTION_TYPE=18" | convert timeformat="%Y%m" ctime(_time) AS month | chart avg(duration), count by month | convert timeformat="%H:%M:%S" ctime(avg(duration))

[CA Expirations Monthly table]
search = index="reporting" source="/splunkdata/reporting_prod_certs" |dedup REQUEST_ID, POLICY_OID | eval expdate=strptime(EXPIRATION_TIME, "%Y-%m-%d %H:%M:%S.%Q") | convert timeformat="%Y%m" ctime(expdate) AS expyearmonth| eval timenow=now() | where expdate > timenow | chart count over CA by expyearmonth where top100 | addtotals |addcoltotals

[CA Expirations Monthly, future]
search = index="reporting" source="/splunkdata/reporting_prod_certs" |dedup REQUEST_ID, POLICY_OID | eval expdate=strptime(EXPIRATION_TIME, "%Y-%m-%d %H:%M:%S.%Q") | convert timeformat="%Y%m" ctime(expdate) AS expyearmonth| eval timenow=now() | where expdate > timenow | stats count by expyearmonth

[SERVICE1 Approver Stats per Month]
search = index="reporting" source="/splunkdata/reporting_prod_requests" earliest=-12mon@mon ACTION_TYPE="18" | timechart COUNT span="1month" by ACTOR_NAME where top100 | convert timeformat="%Y-%m" ctime(_time) AS Month | sort _time | fields - _time | table Month * |addtotals fieldname="total" |eval total = total . " #" |addcoltotals label=total labelfield=Month | eval total = total . " #"

[Signings Per Month from logs]
search = source="*service1*" "checkuseraccess" AND "\"Standard application\" - start" earliest=-12month@month | convert timeformat="%Y-%m" ctime(_time) AS month | chart count by month | fields - _time |fields + month, *

[SERVICE2 PROD Signings per user TOP 10last month]
search = index=service2 source="/service2logs/*" SubmissionStatus=SigningSuccessful SigningType=PROD earliest=-1mon@mon latest=@mon| chart count by ClientUsername | sort - count |head 10

[SERVICE2 Succesful Signings per product / month]
search = index=service2 source="/service2logs/*" SubmissionStatus=SigningSuccessful SigningType=PROD | convert timeformat="%Y-%m" ctime(_time) AS Month | chart count over Month by ProductName WHERE top 100 | fields - _time |fields + Month,*

[SERVICE1 Statuses Per Month]
search = index="reporting" source="/splunkdata/reporting_prod_requests" earliest=-12mon@mon | convert timeformat="%Y-%m" ctime(_time) as "Year-Month"| chart count over Year-Month by ACTION_TYPE WHERE top 100|rename 1 as "Developer submitted", 10 as "Signing failed", 16 as "Package creation OK", 13 as "Request expired", 17 as "Package creation failed", 19 as "Approver denied", 2 as "Developer recalled", 201 as "JIT Enrolling failed", 3 as "Manager accepted" , 4 as "Approved accepted", 11 as "Request deleted", 18 as "Approver accepted", 200 as "JIT enroll succeeded", 9 as "Signing srvr Signing OK", 101 as "REVOCATION_MANAGER_SUBMIT", 12 as "REQUEST_DEAD_LETTER_SIGNING_ACTION", 300 as "CAPABILITY_OWNERS_ACCEPTANCE"

[3rd Party Beta Applications to Manufacturer Domain]
search = "categoryGroup=CategoryGroup[id=362,name=3rd Party Beta Applications to Manufacturer Domain" source="*service1.log"

[Developer submitted request count]
search = index=reporting source="/splunkdata/reporting_prod_requests" ACTION_TYPE=1 | dedup REQUEST_ID | convert timeformat="%Y-%m" ctime(_time) AS Month|chart count by Month where top100

[SERVICE1 Rejected, Failed etc. requests]
search = index=reporting   source="/splunkdata/reporting_prod_requests" ACTION_TYPE=2 OR ACTION_TYPE=4 OR ACTION_TYPE=6 OR ACTION_TYPE=8 OR ACTION_TYPE=10 OR ACTION_TYPE=12 OR ACTION_TYPE=14 OR ACTION_TYPE=15 OR ACTION_TYPE=17 OR ACTION_TYPE=19 OR ACTION_TYPE=201 OR ACTION_TYPE=301 | dedup REQUEST_ID | convert timeformat="%Y-%m" ctime(_time) AS Month| chart count by Month where top100

[SERVICE1 Manual approvals, count]
search = index=reporting source="/splunkdata/reporting_prod_requests" ACTION_TYPE=18 | dedup REQUEST_ID |convert timeformat="%Y-%m" ctime(_time) AS Month|chart count by Month where top100

[SpecialApp Signings per Month]
search = index="reporting" source="*requests" CATEGORY="*SpecialApp*" earliest=-6months@month| dedup REQUEST_ID | convert timeformat="%Y-%m" ctime(_time) AS Month | stats count as "SpecialApp Signings" by Month | table YearMonth *

Appendix 2: Python code for fetching log data from a Servlet

```
#!/usr/bin/python
# REQUIRES: Python >2.6
#
# This script is intended to be run from python interpreter integrated into Splunk.
#
# Uses Servlet connection to fetch data from database and saves the results in
# files. Checks that HTTP server return code is 200 and assumes everything is OK.
#
# File format here is CSV (Comma Separated Values) as this is suitable for
# data fetched from relational database, query result being essentially a table.
#
# Parameter for the servlet is EPOCH millisecond, which is produced from the last
# timestamp in the log file. Servlet then performs pre-defined SQL query
# and returns CSV formatted log rows, which this script then writes to a file.
# This file can then be indexed by Splunk as any log file.
#
from __future__ import with_statement
import urllib
import os
import sys
from time import localtime,strftime
import datetime
import time

def unix_time(dt):
    epoch = datetime.datetime.fromtimestamp(0)
    delta = dt - epoch
    return delta.total_seconds()

def unix_time_millis(dt):
    return unix_time(dt) * 1000.0

def parsetime(line):
    return datetime.datetime.strptime(line,"%Y-%m-%d %H:%M:%S.%f")

def getlasttimestamp(logfile):
            fname = logfile

            with open(fname, "r") as f:
                f.seek(0,2)
                fsize = f.tell()
                f.seek (max (fsize - 1024,0), 0)
                lines = f.readlines()
            lines = lines[-1:]
            substr = lines[0].split(",",1)
            foo = substr[0].replace('"', '').strip()
```

```
            lasttimestamp = parsetime(foo)
        lasttime_epoch = unix_time_millis(lasttimestamp)
        lasttime_int = int(lasttime_epoch)
            return lasttime_int


request_file_path = "/splunkdata/reporting_prod_requests"

time_now = datetime.datetime.now()
now_text = time_now.strftime("%Y-%m-%d %H:%M:%S") # SQL query required format

start_text = "2001-01-01 00:00:00.001"

# Some examples of date conversions. %f first in Python 2.6:

# Epoch time from struct_time: time.mktime(<struct>)
# struct_time from epoch: time.gmtime(<epoch>)
# struct time to string: time.strftime(format, <struct_time>)

# start struct = time.strptime(start text, "%Y-%m-%d %H:%M:%S.%f")
# start_epoch = time.mktime(start_struct)

start_timestamp = parsetime(start_text)
start_epoch = int(unix_time_millis(start_timestamp))


# Default to start_epoch, if the last_requesttime cant be read from the file
last_requesttime = start_epoch
if (os.path.isfile(request_file_path) and os.path.getsize(request_file_path) > 0):
    try:
    last_requesttime = getlasttimestamp(request_file_path)
    except IOError:
        sys.stderr.write("Could not read timestamp from file, using default");
        last_requesttime = start_epoch # Default, just in case
    except ValueError:
        sys.stderr.write("ERROR: Last timestamp format mismatch")
        last_requesttime = start_epoch # Default, just in case
else:
    sys.stderr.write('INFO: ' +request_file_path + ' not found. Starting from zero. \n')
    last_requesttime = start_epoch # Default, just in case

requesturl = "http://Server/Servlet-path/get-requests?param1=" + str(last_requesttime)

data = urllib.urlopen(requesturl)
httpcode = data.getcode()

if httpcode == 200:
    try:
```

```
        # CSV header row. This is STATIC, and written to file only once
         headers = "ACTION_TIME,ACTION_TYPE,ACTOR_ID,STATUS,REQUEST_ID,
                                DEVELOPER_ID,DEVELOPER_NAME,IS_EXTERNAL,WORKFLOW,SUBMIT_DATE,
                    CATEGORY,PLATFORM,ACTOR_NAME,FILES_SIZE_MB"

         if not os.path.exists(request_file_path):

             saveheaders = open(request_file_path, 'w')
             saveheaders.write(headers)
             saveheaders.write("\n")
             saveheaders.close()

         f1=open(request_file_path, 'a+')
         line = data.readline()

         while line:
             f1.write(line)
             # Parse the timestamp from the line
             fieldlist = line.split(',')
             # If there is another line, continue while-loop
             line = data.readline()
         f1.close()
     except IOError, (errno, strerror):
         sys.stderr.write("I/O error(%s): %s" % (errno, strerror))
         sys.exit(2)
else:
     sys.stderr.write("Error opening "  + requesturl + " code " + str(httpcode) + "\n")
     sys.exit(2)

#<END PYTHON CODE>
```

Appendix 3: Java Servlet for querying data from Oracle SQL Database

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.Date;
import java.text.*;
import java.sql.*;
import oracle.jdbc.pool.OracleDataSource;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public final class RequestQuery extends HttpServlet {

		private static final long serialVersionUID = 3L;

		public void doGet(HttpServletRequest request, HttpServletResponse response)
					throws IOException, ServletException
					{
			SimpleDateFormat logTime = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss.SS");
			Date t;
			String dateString;
			// param1 should be EPOCH millisecond date, type LONG - for Splunk reporting
			String queryDate  = request.getParameter("param1");

			if(queryDate == null || queryDate.length() == 0) queryDate = "1262304000000";

			// Sanity check of date input, it should be epoch...
			try{
					t = new Date(Long.parseLong(queryDate));
			}catch (NumberFormatException e){
					throw new ServletException(e);
			}
			dateString = logTime.format(t);

			response.setContentType ("text/plain;charset=utf-8");
			PrintWriter writer = response.getWriter();

			// Make the Request Query and embed date
	 String reportQuery = "SELECT * FROM (SELECT ra.action_time AS ACTION_TIME, ra.action_type AS ACTION_TYPE, ra.user_id AS ACTOR_ID, des.description AS
				STATUS, req.request_id as REQUEST_ID, req.owner_id AS DEVELOPER_ID, req.owner_name AS DEVELOPER_NAME, req.source AS IS_EXTERNAL,
				wf.workflow_name AS WORKFLOW, req.request_date AS SUBMIT_DATE, cg.name AS CATEGORY, type.name AS PLATFORM, nu.username AS ACTOR_NAME, (
				SELECT ROUND(SUM(rf.file_size)/1024/1024, 3) FROM service1.request_file rf WHERE rf.request_id=req.request_id ) AS FILES_SIZE_MB FROM
				service1.request_action ra INNER JOIN service1.request req ON req.request_id=ra.request_id INNER JOIN service1.stattool_request_state_desc
				des ON req.state=des.state INNER JOIN service1.category_group cg ON req.cg_id=cg.cg_id INNER JOIN service1.type type ON
```

```
                type.type_id=cg.type_id LEFT OUTER JOIN service1.workflow wf ON req.workflow_id=wf.workflow_id LEFT OUTER JOIN service1.service1_user nu ON
                ra.user_id=nu.employee_id WHERE ra.action_time > to_timestamp('"+dateString+"','YYYY-MM-DD HH24:MI:SS.FF3') AND ra.action_time <= sysdate
                ORDER BY ra.action_time ASC) WHERE ROWNUM < 10000";

        try {
                                doSplunk(reportQuery, writer);
                } catch (SQLException e) {
                                e.printStackTrace();
                                throw new ServletException("Query failed with date input "+dateString+" : ",e);
                }
                                }

private void printSplunk(ResultSet rset,ResultSetMetaData rsmd, PrintWriter writer) throws SQLException {
                // Print the results for Splunk indexing purposes
                if(rset != null && rsmd != null){
                                int numCols = rsmd.getColumnCount();
                                int currCol = 1;

                                currCol = 1;
                                while (rset.next()) {
                                                if(currCol != 1) writer.print(",");
                                                while (currCol <= numCols) {
                                                                writer.print("\"");
                                                                // Timestamps should be printed with getTimestamp(), as as strings
                                                                // there would be confusing whitespace

      if(rsmd.getColumnTypeName(currCol).equals("TIMESTAMP"))writer.print(rset.getTimestamp(currCol));
                                                                else writer.print(rset.getString(currCol));
                                                                writer.print("\"");
                                                                if(currCol < numCols) writer.print(",");
                                                                currCol++;
                                                }
                                                currCol = 1;
                                                writer.print("\n");
                                }
                }
                else {
                                writer.println("<b>ERROR: ResultSet or ResultSetMetaData is null!</b>");
                }
      }

      private void doSplunk(String query, PrintWriter writer) throws SQLException {
                Connection conn = null;
                Statement stmt = null;
                ResultSet rset = null;
                try {
```

```
                        if (query != null){

                                OracleDataSource ods = new OracleDataSource();

                                ods.setURL(getServletContext().getInitParameter("database.connstring"));
                                ods.setUser(getServletContext().getInitParameter("database.username"));
                                ods.setPassword(getServletContext().getInitParameter("database.password"));
                                conn = ods.getConnection();

                                stmt = conn.createStatement();
                                // Default for Oracle FetchSize is 10, we need bigger
                                stmt.setFetchSize(200);
                                rset = stmt.executeQuery(query);
                                ResultSetMetaData rsmd = rset.getMetaData();

                                // "Raw" txt print for Splunk
                                printSplunk(rset,rsmd, writer);

                                rset.close();
                                rset = null;
                                stmt.close();
                                stmt = null;
                                conn.close();
                                conn = null;
                        }
                        else {
                                writer.println("<b>ERROR: query is null!</b>");
                        }

                }catch(SQLException e) {  throw e; }

                finally{
                        if (rset != null) {
                                try { rset.close(); } catch (SQLException e) {  writer.println("Failed to close rset: "+ e.getMessage());
                        }
                                rset = null;
                        }
                        if (stmt != null) {
                                try { stmt.close(); } catch (SQLException e) { writer.println("Failed to close stmt: "+ e.getMessage()); }
                                stmt = null;
                        }
                        if (conn != null) {
                                try { conn.close(); } catch (SQLException e) { writer.println("Failed to close conn: "+ e.getMessage()); }
                                conn = null;
} } } }
// END OF JAVA CODE
```

Appendix 4: Difference in Splunk and SQL results

(Note: Comma used as a decimal separator)

#	DB	Spl	DIFF	DIFF-%
1	8512	8115	397	4,7%
2	7809	7620	189	2,4%
3	7600	7600	0	0,0%
4	7155	7146	9	0,1%
5	7096	7086	10	0,1%
6	7061	7060	1	0,0%
7	7010	6725	285	4,1%
8	6716	6616	100	1,5%
9	6577	6578	-1	0,0%
10	6498	6493	5	0,1%
11	6468	6337	131	2,0%
12	6369	6278	91	1,4%
13	6362	6229	133	2,1%
14	6341	6181	160	2,5%
15	6264	6178	86	1,4%
16	6231	6170	61	1,0%
17	6178	6151	27	0,4%
18	6153	6119	34	0,6%
19	6105	5849	256	4,2%
20	5991	5820	171	2,9%
21	5937	5743	194	3,3%
22	5820	5721	99	1,7%
23	5685	5593	92	1,6%
24	5682	5581	101	1,8%
25	5677	5544	133	2,3%
26	5599	5522	77	1,4%
27	5584	5516	68	1,2%
28	5568	5511	57	1,0%
29	5545	5470	75	1,4%
30	5526	5455	71	1,3%
31	5516	5444	72	1,3%
32	5506	5382	124	2,3%
33	5473	5349	124	2,3%
34	5463	5252	211	3,9%
35	5352	5215	137	2,6%
36	5247	5167	80	1,5%
37	5167	5086	81	1,6%
38	5157	5066	91	1,8%
39	5141	5055	86	1,7%
40	5105	4917	188	3,7%
41	5066	4889	177	3,5%
42	5023	4882	141	2,8%
43	5001	4845	156	3,1%
44	4887	4831	56	1,1%
45	4865	4790	75	1,5%
46	4853	4786	67	1,4%
47	4846	4771	75	1,5%
48	4838	4754	84	1,7%
49	4724	4715	9	0,2%
50	4691	4686	5	0,1%
51	4686	4672	14	0,3%
52	4680	4546	134	2,9%
53	4606	4508	98	2,1%
54	4573	4506	67	1,5%
55	4557	4502	55	1,2%
56	4531	4457	74	1,6%
57	4506	4456	50	1,1%
58	4503	4394	109	2,4%
59	4465	4380	85	1,9%

60	4405	4319	86	2,0%
61	4385	4305	80	1,8%
62	4336	4303	33	0,8%
63	4320	4276	44	1,0%
64	4319	4271	48	1,1%
65	4303	4229	74	1,7%
66	4298	4228	70	1,6%
67	4287	4217	70	1,6%
68	4273	4217	56	1,3%
69	4273	4179	94	2,2%
70	4242	4172	70	1,7%
71	4240	4156	84	2,0%
72	4235	4153	82	1,9%
73	4220	4150	70	1,7%
74	4217	4135	82	1,9%
75	4213	4124	89	2,1%
76	4184	4103	81	1,9%

77	4178	4085	93	2,2%
78	4177	4063	114	2,7%
79	4169	4045	124	3,0%
80	4156	4037	119	2,9%
81	4156	4037	119	2,9%
82	4150	4028	122	2,9%
83	4134	4020	114	2,8%
84	4108	4005	103	2,5%
85	4091	3992	99	2,4%
86	4080	3979	101	2,5%
87	4067	3963	104	2,6%
88	4061	3956	105	2,6%
89	4037	3939	98	2,4%
90	4028	3929	99	2,5%
91	4021	3877	144	3,6%
92	3964	3867	97	2,4%
93	3963	3867	96	2,4%

94	3914	3844	70	1,8%
95	3900	3823	77	2,0%
96	3883	3818	65	1,7%
97	3877	3800	77	2,0%
98	3874	3791	83	2,1%
99	3868	3787	81	2,1%
100	3858	3756	102	2,6%

Average DIFF: 1.9